Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería en Electrónica y Automática Industrial

Trabajo Fin de Grado

Multi-Object Tracking System based on LiDAR and RADAR for Intelligent Vehicles applications

Autor: Santiago Montiel Marín

Tutores: Luis Miguel Bergasa Pascual y Carlos Gómez Huélamo

2021

UNIVERSIDAD DE ALCALÁ

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería en Electrónica y Automática Industrial

Trabajo Fin de Grado

Multi-Object Tracking System based on LiDAR and RADAR for Intelligent Vehicles applications

Autor: Santiago Montiel Marín

Directores: Luis Miguel Bergasa Pascual y Carlos Gómez Huélamo

Tribunal:

Presidente: Felipe Espinosa Zapata

Vocal 1º: Rafael Barea Navarro

Vocal 2º: Luis Miguel Bergasa Pascual

Calificación: .....

Fecha: .....

To my beloved parents, for without them it would have been impossible to…

"The most noblest pleasure is the joy of understanding."
Leonardo da Vinci

Acknowledgements

It is not flesh and blood, but the heart, that makes us parents and children.

Friedrich Schiller

Este trabajo es el fruto de una búsqueda que ha durado cuatro largos e intensos años. Años de muchas horas de trabajo en los que dicha búsqueda se ha traducido en una transformación, en la que comenzaba como un adolescente recién llegado a la Universidad y he terminado como un joven que ha encontrado su vocación profesional y la dirección hacia la que quiere encomendar sus futuras líneas de trabajo.

Primeramente me gustaría agradecer tanto a Luis Miguel Bergasa Pascual, tutor de este trabajo, como a Carlos Gómez-Huélamo, amigo y cotutor del mismo, la confianza depositada en mí y la inversión que han hecho para mostrarme un camino profesional el cual tengo ganas de recorrer con ambición. La investigación en el campo de la conducción autónoma se ha convertido en la materia en la que me gustaría profundizar y desarrollarme como profesional.

Después quería recordar a todas aquellas personas que me han acompañado durante este periplo en mi primera etapa en la Universidad. A todos los amigos que he conocido y a mis compañeros y profesores del Equipo de Robótica, que se han convertido en un apoyo incondicional e indispensable. No podría cerrar este párrafo sin mencionar a mis compañeros de batalla: Ana Belén, César, los Javis, Fabio y Navil, y al mentor más extraordinario que se puede tener, el profesor Ángel Álvarez.

Por último, a mi familia. Padres, abuelos, tíos y primos por los que siento admiración y que han sido uno de los pilares sobre los que me apoyaba para ser capaz de completar este camino. Dedicatoria especial a mis padres, Santiago y María Dolores. Ellos dos han sido un motor de ilusión, esperanza y ánimo durante todos estos años y les debo estar eternamente agradecido por ello. Como dice la dedicatoria de este libro: "a mis amados padres, porque sin ellos hubiera sido imposible...".

Resumen

El presente Trabajo Fin de Grado tiene como objetivo el desarrollo de un Sistema de Detección y Multi-Object Tracking 3D basado en la fusión sensorial de LiDAR y RADAR para aplicaciones de conducción autónoma basándose en algoritmos tradicionales de Machine Learning. La implementación realizada está basada en Python, ROS y cumple requerimientos de tiempo real. En la etapa de detección de objetos se utiliza el algoritmo de segmentación del plano RANSAC, para una posterior extracción de Bounding Boxes mediante DBSCAN. Una Late Sensor Fusion mediante Intersection over Union 3D y un sistema de tracking BEV-SORT completan la arquitectura propuesta.

Palabras clave: Conducción autónoma, fusión sensorial, detección de objetos, seguimiento de múltiples objetos, aprendizaje automático.

Abstract

This Final Degree Project aims to develop a 3D Multi-Object Tracking and Detection System based on the Sensor Fusion of LiDAR and RADAR for autonomous driving applications based on traditional Machine Learning algorithms. The implementation is based on Python, ROS and complies with real-time requirements. In the Object Detection stage, the RANSAC plane segmentation algorithm is used, for a subsequent extraction of Bounding Boxes using DBSCAN. A Late Sensor Fusion using Intersection over Union 3D and a BEV-SORT tracking system complete the proposed architecture.

Keywords: Autonomous Driving, Sensor Fusion, Object Detection, Multi-Object Tracking, Machine Learning.

Extended Abstract

Autonomous driving has undergone a major push in recent years thanks to the latest advances in both hardware, with sensors offering great performance, and software, in which Machine and Deep Learning techniques have led to the birth of architectures capable of understanding the world and making safe decisions to navigate it. This, together with the electrification of vehicles and research into new fuels, is set to bring about a revolution and change the mobility paradigm we know today towards a more efficient, safer and more environmentally aware model.

Therefore, this Final Degree Project is framed within the perception layer of an autonomous driving system. Its main objective is the development of a 3D Object Detection and Multiple Tracking System with the premise of understanding the environment surrounding the vehicle from the data received from LiDAR and RADAR sensors.

To achieve the objective, use will be made of state-of-the-art technologies such as CARLA Simulator and ROS software development environment for robots, which are used to build software applications with real-time requirements written in Python programming language. Cutting-edge technologies in the field of software engineering are also used for version control and containerisation, such as Git and Docker, respectively.

In terms of the techniques used, an architecture is proposed that combines classical algorithms with Machine Learning algorithms to obtain an efficient, fast and simple system. The RANSAC algorithm is used for plane segmentation tasks, which precedes an object detection phase in which a Machine Learning-based approach is provided using the DBSCAN unsupervised clustering algorithm. Afterwards, a data association and sensory fusion is performed between the detections of both sensors using an Intersection over Union 3D algorithm that serves as input for a SORT-based algorithm that is based on Linear Kalman Filter and Hungarian Algorithm and is in charge of tracking the detections and performing an over-time follow-up.

In addition, the work is accompanied by extensive studies in the field, ranging from the physical fundamentals governing LiDAR and RADAR sensors to reviews of the architectures proposed by researchers in the state of the art in order to achieve a complete contextualisation and understanding of the problem of systems for perception in autonomous driving.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

There is no easy way from Earth to the stars.

Lucius Annaeus Seneca, the Younger

1.1 Motivation

In recent years, academia and industry have both shown great interest in the development of autonomous vehicles (AV) and self-driving cars as they are seen as the main characters of a booming sector that marks the next transportation revolution. Taking into account all the advances made in recent years and knowing that the automotive sector wants to evolve to a level of fully autonomous driving, where the intervention of a human driver is null and void, it can be concluded that research in this field is a sector that is completely in vogue.

Autonomous vehicles are defined as intelligent transportation systems able to drive safely with little or no human input. In order to afford this task, they are capable of sensing its environment and extracting the most important features of the scene that will make them take the most appropriate decisions for each circumstance. It is equivalent to saying that an autonomous vehicle is one capable of performing the functions related to the five domains or layers present in any autonomous vehicle architecture: perception, localisation, mapping, decision making and control.

A fully autonomous architecture is still a project several years away. Mainly because of the technical difficulties involved, but also because of the social and legal difficulties. No company or industrial organisation has presented a ratified methodology for level 4 or 5 vehicles, but the community has issued a maxim to follow: simulation is essential and critical to build safe autonomous vehicles, so it is necessary to advance in the field of vehicle simulation with an architecture composed of sensors of high computational complexity. This is where Artificial Intelligence, Machine and Deep Learning take on an important role for future work, getting involved in tasks such as detecting the most relevant objects around the vehicle or assessing the situation to make the safest decisions while driving.

Given the complexity, breadth and transversality of the problem of autonomous driving, this work is framed within one of the layers of the architecture of an autonomous vehicle: perception. This is the layer in charge of processing data from sensors for subsequent construction and understanding of the environment. The most commonly used sensors are currently camera-type sensors, LiDAR, RADAR, ultrasound, GPS, mechanical odometry and inertial measurement units. After that, data from all sensors in the vehicle are processed by sensor fusion techniques to get a complete understanding of the scene.

To justify all this investment and development, arguments can be made for an increase in driving efficiency when the human factor is removed, as well as for an increase in road safety brought about by the removal of factors that negatively affect the driver's understanding and perception of the environment, such as fatigue, inattention or the consumption of substances such as alcohol or drugs. Other factors not directly related to driving but influencing this boom in the sector could also be highlighted, such as improvements in energy efficiency, society's shift towards a more ecological and environmentally conscious model, and the ageing of the population living in urban environments.

Therefore, we can elaborate on some of the main advantages of the development of autonomous vehicles, which are the following:

- **Reduction in the number of accidents**: several reports state that the majority of accidents are caused by human error. Factors such as dangerous overtaking, speeding or the elimination of the negative circumstances present in drivers, such as those mentioned above, are totally removed from the scene.
- **Mobility for all**: autonomous driving can influence the creation of a more inclusive modern society. Certain sectors of populations, such as elderly people, blind people or people with reduced mobility, could have access to an efficient transportation system with which they would have no problems in getting to their destination without any external support.
- **Committed to the environment**: an autonomous driving car would adapt its driving style according to the road conditions assisted by intelligent systems. This means that the driving will be more efficient than the one performed by a human, so it will mean a decrease in energy consumption. Also, it is important to remark that autonomous driving revolution is accompanied by a energy car revolution. Cars will drive better and powered by cleaner forms of energies, such as electricity from renewable sources or hydrogen.
- **Transport facilitation**: A more efficient and adaptable driving means less frequent traffic jams and fewer events that slow down the road. Moreover, driver rest periods would not be needed so the time for transporting goods would be significantly reduced.

Therefore, it is not unreasonable for the reader to think that the inclusion of the autonomous car poses a complete paradigm shift. What would you do in the car if you didn't have to worry about driving? Would you again suffer from time incompatibilities due to having to pick up your children from school? Would you go back to looking for parking for many minutes or would you let the car do it autonomously and let you focus on your tasks?

1.2 Historical context

1.2.1 1920-1970: Early origins and first experiments

The first attempts to automate driving were made in the 1920s in the United States. Firstly, *Houdina Radio Control*, a radio equipment company, showed its invention "American Wonder". It was a radio-controlled car that circulated through the streets of New York City in 1925. This vehicle was operated by an employee of the company in another car that followed it. It can be seen as the first attempt to automate the direction and movement of the vehicle. The following attempt occurred in Milwaukee in 1926. In this second case, Achen Motor, a distribution car company, showed the world its "Phantom Auto" that can be seen in Figure 1.1.

It is important to remark that these first experiments implemented an electronic circuit that allowed a remote control. Moreover, this can not be considered as an autonomous vehicle per se. In fact, it is reported another demonstration in 1932 in which the car was operated by a person in a low-flying airplane. These cars were called as "one of the most amazing products of modern science" and the public was captivated.

Figure 1.1: 1935 Pontiac, model used as Phantom Auto
Source. theatlantic.com. Your Grandmother's Driverless Car

Another remarkable experiment was the one done by Tsukuba Mechanical Engineering Laboratory in 1977. This Japanese company developed the first semi-autonomous vehicle and it was able to understand the scene that surrounded the vehicle in specially marked streets. These signals or marks were interpreted by a vision system composed by two cameras and an analogous computer mounted in the vehicle.

1.2.2 1980-1990: Academia leads breakthroughs

During the following years, more experiments and demonstractions were performed and Japan, the United Kingdom and Europe joined this field of research. Nevertheless, it was not until 1980s when considerable advances were made. For example, Mercedez-Benz and Bundeswehr University Munich developed a vision-guided van that achieved 63 km/h on streets without traffic.

In the United States, it did not take long to DARPA, a research centre associated with U.S. Department of Defense, to show the world its Autonomous Land Driven Vehicle (ALV) project in 1986. This work, done collaboratively with top universities such as Carnegie Mellon University (UCM), resulted in the first autonomous architecture that made use of LiDAR, computer vision and robotic control, achieving 31 km/h.

In view of satisfactory results, CMU started its project Navlab. It was a series of autonomous and semi-autonomous vehicles developed at The Robotics Institute of this university. These vehicles were numbered from 1 to 10. Navlab 1 was built in 1986 using a Chevrolet van that included different hardware stations and it obtained similar results to ADV project. In 1990, Navlab 2 was built in top of a U.S. Army Hummer vehicle. It has two different processing units, three Sparc 10 computers for high level processing and two computers based on Motorola 68000 processors for low level control. Due to its heavy characteristics, it became the first autonomous vehicle to drive autonomously through rough terrains, achieving 10 km/h. But when it came to road, Navlab 2 was able to drive at 110 km/h. These two vehicles were capable of controlling the steering wheel autonomously and estimating its position thanks to a inertial navigation system.

The most important victory came with Navlab 5. In July 1995, using an adapted Pontiac Trans Sport minivan, the CMU team was able to perform a long travel in which the vehicle drove autonomously through the 4500 km that separate Pittsburgh and San Diego. During the journey, the vehicle averaged over 100 km/h. Thanks to this impressive achievement, Navlab 5 was added to Robot Hall of Fame in 2007. The rest of the Navlab vehicles were also exploring automation for public transportation with buses or vans.

On the other side of the Pacific Ocean but also in 1990s, South-Korean professor Han Min-Houng worked on a self-driving car that was tested in Seoul and Busan. His project accumulated a total of 17 kilometres travelled around the Korean capital and performed a travel between both cities. Despite its promising results, the Korean economy was focused on other sectors and the funding was cut. Some newspapers considered Prof. Min-Houng as "a man ahead of its time for his country, an Elon Musk of his era".

Meanwhile, the PROMETHEUS project was born in Europe, a programme in charge of encourage the efficiency and road safety in a magnitude never seen before. Consequently, Daimler-Benz continued working and improving its autonomous driving architectures. In 1994, they developed two "twin robot vehicles", as they defined. They drove more than 1000 km on a French three-lane highway. Among its main skills, these cars were able to cope with autonomous driving in free lanes, convoy driving, and lane changes with autonomous passing of other cars.

The cherry on the top of PROMETHEUS came with VITA, an abbreviation of *Vision Information Technology Application*. This system was composed by small video cameras installed behind the windscreen and rear window that enabled a steering of the vehicle using computer vision. Thanks to this advanced system, Mercedes-Benz adapted an S-Class (shown in Figure 1.2) that performed a travel between Munich and Copenhague and back, covering a distance of 1600 km. It reached 175 km/h in Autobahn highway, the German highway without speed limits. It drove in traffic and performed overtaking manoeuvres without human intervention.

Figure 1.2: Mercedes-Benz VITA2 prototype in its journey Berlin-Copenhague
Source. autoevolution.com. A Short History of Mercedes-Benz Autonomous Driving

Other substantial advances were made in that year but in the United Kingdom. The UK Department of Trade and Industry joined forces with Jaguar Cars. They both developed parts for a semi-autonomous car. Also in Europe, there were projects in Italy (*Mile Miglia in Automatico* by University of Parma) and in The Netherlands (automated people mover at Schiphol Airport).

In the late 1990s, automotive companies such as Toyota put their research focus on driving automation. In May 1998, this company became the first to introduce an Adaptive Cruise Control (ACC) system on a production luxury vehicle which was sold in Japan.

1.2.3 2000s: A decade of challenges

During this decade, autonomous driving experimented substantial advances due to the celebration of many challenges. Most of them were hosted and encouraged by DARPA. This organization took a leading role with the celebration of DARPA Grand Challenge in 2004. The U.S. Congress authorized DARPA to offer a prize of 1$ million to the winner of this challenge which consisted in a 240 km race for driverless cars through Mojave Desert in the Southwest of the United States. None of the vehicles finished the route, although the CMU vehicle Sandstorm obtained the best score, with 12 kilometres covered. As there was not a winner, a second DARPA Grand Challenge was scheduled for the following year, 2005.

The second edition of the competition took place in the same scenario. 23 teams accepted the challenge of crossing the arid desert autonomously. The advances made during that year were really impressive as all but one surpassed the mark obtained by CMU the previous year. Finally, 5 teams completed successfully the race and there was a champion: Stanford University. Stanford Racing Team, which was led by Professor Sebastian Thrun, assembled an autonomous architecture on a Volkswagen Touareg denominated Stanley that can be seen in Figure 1.3.

Figure 1.3: Stanley vehicle parked after the 2005 DARPA Grand Challenge
Source. en.wikipedia.org. Stanley (vehicle) from Wikipedia, The Free Encyclopedia.

Stanley used 5 roof-mounted LiDAR sensors and a multi-camera system for perceiving its surrounding scene. This is striking since all the previously mentioned architectures based their perception of the environment solely on computer vision. These sensors were used to detect obstacles based on a machine learning approach and to build a 3D map of the scene which helped GPS and other sensors to cope with localization tasks. It was also implemented a drive-by-wire control system that allowed to perform steering tasks taking into account vehicles variables such as speed, direction and decision making. Stanford University stated that more than 100.000 lines of code were developed to achieve these results and, in reward for its victory, the university received a 2$ million prize.

DARPA considered that crossing the desert was proof enough that autonomous navigation in friendly environments was possible, so it decided to increase the difficulty of the game by taking the third edition of the challenge to urban environments. DARPA Urban Challenge took place on November 2017 at Georgia Air Force Base in California. This time, participants should follow the Californian driving laws and cope with different use cases in a closed loop circuit during 6 hours. It is possible to say that in this edition the main difficulties came lay in software because vehicles must take decisions in real time based on the conditions of the scenario, obeying a set of rules. Tartan Racing became champion. This team was based at Carnegie Mellon University and was led by Professor Red Whittaker.

These two challenges made both Stanford University and Carnegie Mellon University, including their respective professors, Sebastian Thrun and Red Whittaker, legends and landmarks in the field of autonomous vehicle research, whose recognition continues to this day. As an anecdote, during the DARPA Challenges, most of the participants mounted LiDAR prototypes of the Velodyne brand, which would eventually become one of the most prestigious companies in the world that commercialised this sensor.

1.2.4 2010s: Big automotive companies lead the game

This decade began with the great feat of the University of Parma. Between July and October 2010, the VisLab Research Group made a 15900 km journey from Parma, Italy to Shanghai, China that marked the first intercontinental journey made by an autonomous vehicle. This research group, led by Professor Alberto Broggi, established itself as one of the pioneers of the sector in Europe, reaching a level that could only be replicated by Mercedes-Benz on this continent.

Another university that stood out during the 2010s was Karlsruhe Institute of Technology (KIT). It can be deduced that the strong German automotive industry generates a perfect ecosystem in which the KIT was able to develop perception systems based on stereo cameras in 2013. In the same year they also launched the famous KITTI [1] dataset with which a large number of companies and universities test their software models and solutions in which they are ranked and compared with the rest of the world.

However, this decade is known for other major players entering the scene. Major technology and automotive companies began to make substantial breakthroughs that meant that academia was no longer the only source of innovation and progress.

The first player we will talk about is Google. The big tech company started its journey into autonomous vehicle research at the end of 2009. It hired top engineers such as Sebastian Thrun and Anthony Levandowski and provided them with a fleet of 100 Toyota Prius vehicles with which to develop new solutions. The company's first major victory came in May 2012. In that month, an autonomous car modified with Google's driverless technology had to be tested for a driver's licence under Nevada state law. The test was passed and became the first driver's licence obtained by an autonomous vehicle.

The company has also introduced prototypes of autonomous cars without steering wheels (Figure 1.4) and its fleet of vehicles has travelled thousands of kilometres in many US states. Its meteoric rise would only continue, leading to a rebranding in 2016, after which Google renamed its autonomous car division Waymo, taking on greater independence and resources. Cities such as San Francisco in California, Las Vegas in Nevada or Austin in Texas witness Google's fleet of vehicles taking data every day to further improve the training of their Deep Learning models and decision-making. It is also important to remark that Waymo began testing autonomous minivans without a safety driver on public roads in Chandler, Arizona, in October 2017 and the project continues to go from strength to strength as it raises larger and larger rounds of funding, which are backed by impressive results.

Figure 1.4: Google Firefly autonomous vehicle had no steering wheel.
Source. waymo.com. Our History, Waymo.

Of course, it is essential to talk about Elon Musk when talking about autonomous cars. The South African entrepreneur is one of the most celebrated figures on the world stage in terms of autonomous driving. Musk founded Tesla Motors in 2003 with other partners, but the company's fame came with the development of the Autopilot system. Tesla cars are equipped with an eight-camera system that provides a 360-degree view around the vehicle with a range of 250 metres. Postulated as detractors of LiDAR and RADAR sensors, they rely on their skills and infrastructure to develop models and products that provide a complete solution capable of navigating even in adverse weather conditions, based solely on cameras.

The company affirms that all new Tesla cars have the hardware that will be needed in the future for fully automatic driving in most situations so they can continuously develop the system and provide their users with software updates that add new features to the system in short periods of time, without the need to spend money on new hardware. For now, the launch of the Autopilot system is seen as one of the greatest success stories of the decade.

Also in April 2015, a prototype designed by Delphi Automotive became the first autonomous vehicle to complete a coast-to-coast journey across North America. It travelled from San Francisco to New York, under automatic control for 99% of the total journey.

But the 2010s also brought the first fatal accident involving an autonomous car. A Volvo XC90 autonomous prototype from Uber Advanced Technologies hit Elaine Herzberg, a 49-year-old woman, while she was crossing the street in Arizona. The police investigation concluded that the incident was entirely avoidable, as the safety driver on board the vehicle ignored the signal to switch the car to manual mode in case of emergency measures.

1.2.5 2020s: The decade in which Level 5 will be reached (?)

Thus, it seems that a perfect ecosystem for the development of autonomous vehicles has been created for this decade. According to the National Highway Traffic Safety Administration (NHTSA), vehicles with fully automated safety features and fully autonomous driving could be on the road for a large proportion of vehicles on the highway this decade. Will we be able to complete this great engineering challenge?

1.3 Levels of automation

Given the complexity of the autonomous driving problem, it was necessary to formalise the degree of autonomy of vehicles. A technology whose development has been going on for decades and is constantly advancing, it is difficult to pigeonhole in order to provide legal and economic frameworks with which to deal with this new reality.

The Society of Automotive Engineers therefore set to work in 2014 to provide a 6-level classification system (shown in Figure 1.5), known as J3016, *Taxonomy and Definitions for Terms Related to On-Road Motor Vehicle Automated Driving Systems*.

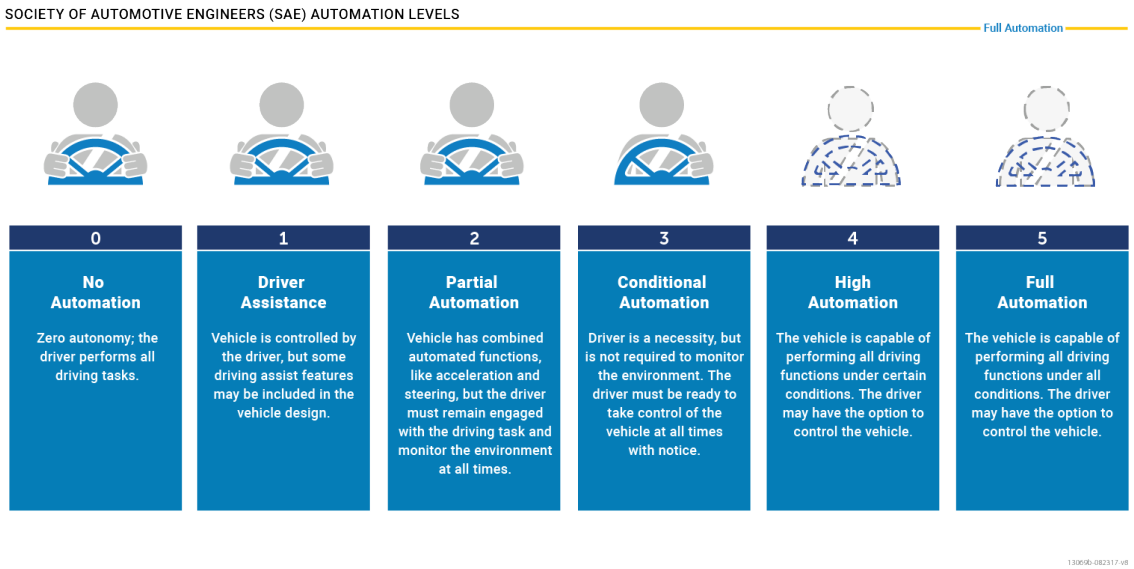


Figure 1.5: SAE J3016 Automation Levels diagram
Source. nhtsa.gov. The Evolution of Automated Safety Technologies.

This classification system, which numbers the levels from 0 to 5, bases its division of levels on the interaction performed by a human as the driver of the vehicle, from an anthropocentric point of view and being inherent to the functionalities implemented in the vehicle. These levels begin at Level 0, where a driver is required to perform full driving to Level 5, in which the driver and the steering wheel are optional elements because the vehicle operates independently in fully autonomous mode.

Human driver monitors the driving environment:

- **Level 0. No Automation**

At this level, the human does all the driving. The presence of a human driver is required to perform all driving-related tasks, even when some tasks are aided by warning or intervention mechanisms such as Emergency Braking System (ABS). Nowadays, the majority of active on-road vehicles are at this level.

- **Level 1. Driver Assistance: hands on**

Although the vehicle continues to be controlled by a human driver at this level, some functionalities are included that offer limited support for certain driving tasks. This is where the concept of ADAS arises: *Advanced Driver Assistance Systems*. This assistance is defined as vehicle actions on either the steering wheel or the accelerator-brake, but not on both mechanisms simultaneously.

An example of technology at this level is Adaptive Cruise Control (ACC), which allows the vehicle to keep a certain distance from the vehicle in front of it by autonomously adjusting the vehicle's speed.

- **Level 2. Partial Automation: hands off**

This is the first level where there may be circumstances where the driver does not need to put his hands on the steering wheel of the vehicle. We can say that we have reached this level when there is a system in our vehicle capable of managing both steering and acceleration or braking.

Tesla cars are said to be framed within this level, although the company says that their cars are able to cope with Level 5 situations with the hardware they have already installed. They will go increasing these Levels with subsequent software updates. Nowadays, Autopilot system allow the vehicle to drive autonomously in highway, only requiring the driver to keep his hands on the steering wheel for a quick transition of control if necessary.

Automated driving system monitors the driving environment:

- **Level 3. Conditional Automation: eyes off**

At level 3, we move from ADAS to Automated Driving Systems (ADS) and we move to a scenario where the user is no longer the most important element of the vehicle. It is therefore a leap in level that may seem at first glance simpler than it really is, because in order for the vehicle to start making most decisions autonomously, it must have highly developed software layers. All this is accompanied by a perfect understanding of the scene through perception systems and good decision making.

In this scenario, the user can take his or her attention away from driving and perform other tasks without being reckless at the wheel. However, full independence has not yet been achieved. There is a set of emergency tasks where the user's attention and decision making is required, which should be specified by the vehicle manufacturer.

Arguably, this is where most of the research is at the moment. Research is being carried out in the world of work, academia and governmental bodies to achieve Level 3 in the very near future. Early attempts at Level 3 vehicles are being made by Audi in its latest version of the A8 model.

- **Level 4. High Automation: mind off**

The main difference between Level 3 and Level 4 is that the vehicle can take over the controls in the situations mentioned in Level 3 if it thinks that its decision making will be better than that of the user. It would do this by activating safety and emergency response mechanisms or protocols.

For example, the car could make a decision to park in a safe place in an adverse situation if the user is not in a position to take the controls at that moment. Right now, current Level 4 vehicles are very limited in terms of legislation and only a few prototypes have been shown to the world by technology giants such as Waymo, NAVYA or Magna.

- **Level 5. Full Automation: steering wheel optional**

At this level, human driving nor attention is not required at all. In fact, steering wheel or pedals are not necessary for a normal vehicle operating mode. This vehicles will be able to work in all kind of surfaces, all over the world and in all conditions. The vehicle will be able to drive under all roadway and environmental conditions that are now managed by a human driver. In fact, human occupants are just passengers and need never be involved in driving.

This Level will take to the maximum exponent all the advantages mentioned in Section 1.1: maximum efficiency and energetic saving, minimum crashes, and open to the whole world. This is our future.

1.4 Overview and structure of the document

As previously mentioned, the work will be framed in the perception layer of an autonomous vehicle and will consist of the development of a multi-object detection and tracking software solution implemented in Python programming language. For this purpose, traditional Machine Learning algorithms will be used, such as the DBSCAN clustering algorithm, through the use of leading libraries in the market such as Open3D or Scikit-Learn. The work can therefore be broken down into the following objectives:

- Study of the theoretical foundations of the operation of LiDAR and RADAR perception sensors.
- Study and training on traditional techniques for LiDAR, RADAR and sensor fusion processing for object detection and tracking.
- Study and mastery of advanced features of Python programming language for a successful realisation of the project.
- Introduction to and mastery of CARLA autonomous driving simulator and Robot Operating System (ROS).
- Contextualisation of the work on the autonomous driving architecture of the Techs4AgeCar project.

In order to achieve the objectives that has been previously mentioned, a software project will be developed and all the material studied during its duration will be collected in this document, whose organisation will be as follows.

- **Chapter 1. Introduction**. This chapter has served to introduce the reader and the author to the problem of autonomous driving, attacking it from different perspectives: historical, economic and social. Key concepts such as levels of automation have been introduced and the chapter has concluded with a presentation of the structure of the document and the objectives of this work.
- **Chapter 2. Techs4AgeCar Project**. *Chapter 2* will serve to put into context and introduce the reader to the research group and project in which this work is framed. Therefore, both the RobeSafe research group and its framework will be presented, paying special attention to the *Techs4AgeCar* project, focused on the development of autonomous driving solutions that help an older segment of the population, together with its predecessor, the *SmartElderlyCar* project.
- **Chapter 3. State of the Art**. In this chapter, a review of the techniques used in the perception layer for autonomous vehicles will be carried out as the present work is framed in this layer. It will serve to study all approaches to the problem of scene perception, supported by a subsequent review of current literature that attempts to mimic the work being done in the research publications.
- **Chapter 4. Theoretical Study**. At this point, the reader will be fully contextualised and immersed in the problem of perception for autonomous vehicles. Therefore, on the one hand, the theoretical foundations of the sensors for perception used during the work, LiDAR and RADAR, will be studied, while on the other hand, the roles taken by each of them in a cooperation through Sensor Fusion are presented.
- **Chapter 5. Software Technologies and Frameworks**. In this chapter, the main characteristics of the software technologies and frameworks in which this work will be developed

will be presented. Therefore, the fundamentals of the Robot Operating System (ROS) will be explained, as well as the advantages of using technologies such as CARLA Simulator, GitHub or Docker.

- **Chapter 6. Work Development**. The development of the work carried out will be presented in *Chapter 6*. From a first beginning in which the architecture for vehicle detection and tracking is proposed, to the final product, including a qualitative analysis of the results obtained.

- **Chapter 7. Conclusions and Future Works**. To conclude this book, an a posteriori analysis of the work carried out will be made, drawing the pertinent conclusions and outlining the lines of future work with which this perception software project can continue to improve.

Chapter 2

Techs4AgeCar Project

Small things flourish by concord. Unity makes strength.

Gaius Sallustius Crispus

2.1 Introduction

The development of this work is framed within the architecture of *Techs4AgeCar* (2019 – 2021), a research project developed by RobeSafe Research Group at Universidad de Alcalá. Funded by Ministerio de Ciencia e Innovación (Spanish Government), the project's main objective is the development of an autonomous electric prototype car, shown in Figure 2.1, capable of assisting senior drivers with different levels of automation.

This project was born as a continuation of *SmartElderlyCar* (2016 – 2018), a project developed jointly by RobeSafe Research Group (Universidad de Alcalá) and GROBIS Research Group (Universidad de Vigo), in which both teams began the development of the mentioned autonomous prototype.

Both projects are aligned with RobeSafe Research Group's main lines of research. The group is developing 3D scene perception techniques by applying sensor fusion techniques that combine information acquired by LiDAR, RADAR and camera sensors. In addition, localisation, navigation and mapping techniques are developed based on the information collected by the sensors on board the group's autonomous platform.

RobeSafe Research Group is a group with a high annual scientific production and with a certain business dimension, as it combines both types of activities, research and business projects with important partners in Europe.

Techs4AgeCar project appears in a context in which, according to *Organization for Economic Co-operation and Development* (OECD), the 25% of the global population will surpass 65 years by 2050. Consequently, the number of elderly drivers will increase in a proportional way. In Spain, senior drivers represent 14% of the total driver population and it is expected to increase until 33% by 2030 [2].

Then, *Techs4AgeCar*'s main proposal is to offer solutions to this problem, combating the rising ageing of population giving freedom and independence to this population group by developing a complete autonomous driving architecture (diagram shown in Figure 2.2), whose main objectives are the following ones:

- Conduct research into technologies for the development of an electric autonomous car to assist the elderly in urban environments.
- Development and implementation of a perception system based on sensory fusion using LiDAR, camera and DGPS technologies.
- To achieve a user-specified point-to-point navigation system through the development of planning and control algorithms.
- Develop algorithms to allow the mapping of the environment, the semantic segmentation of the scene and the real-time localization and detection of obstacles in the path.
- Validation of the previously mentioned algorithms and systems on simulation using CARLA Simulator and on the real prototype at University of Alcalá.

Figure 2.1: *Techs4AgeCar* autonomous vehicle
Source. Demo Techs4AgeCar Project, RobeSafe Research Group

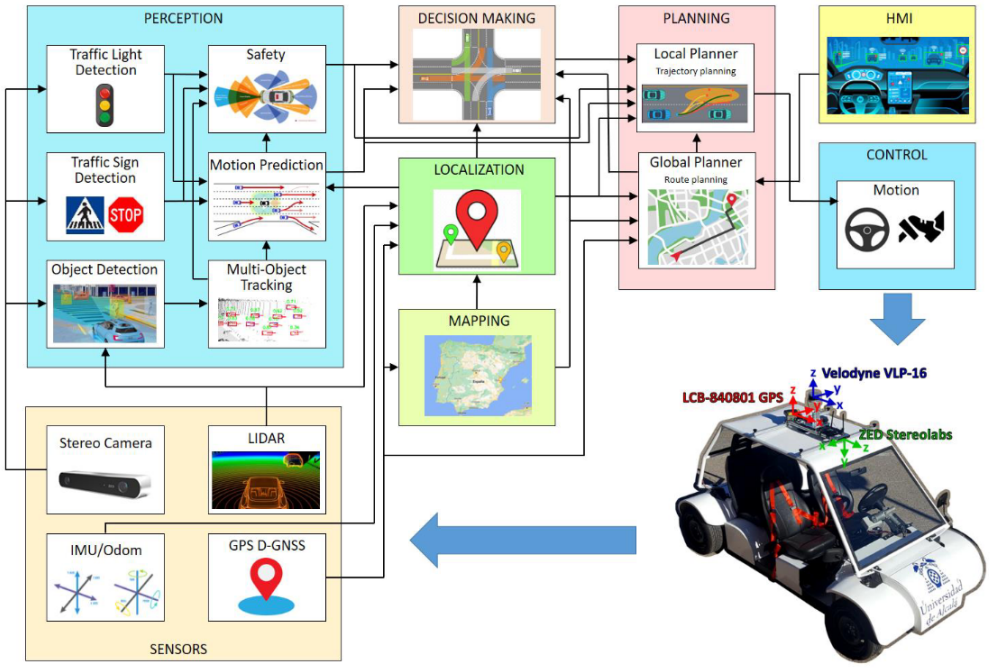


Figure 2.2: *Techs4AgeCar* architecture diagram
Source. RobeSafe Research Group

2.2 Drive-by-Wire Layer

One of the main objectives of the project was the design, modelling and construction of a robust Drive-by-Wire system [3]. The correct functioning of this layer is of vital importance for the development of the project, since if this layer did not work, there would be no autonomous platform on which to develop the rest of the layers.

This system automates and electrifies the transmission of signals from the steering wheel or accelerator and brake pedals to the wheels or other actuators. This allows an intelligent system to make the appropriate decisions in terms of turning and speed to send the commands following an Ackermann geometry model. In this case, it is the Control Layer, which will be explained later, that is responsible for sending these commands to travel the desired trajectory.

2.2.1 Real Prototype

This Drive-by-Wire system was implemented in the real prototype of the project. In 2017, RobeSafe Research Group decided to purchase the open-source TABBY EVO chassis from Open Motors. Thanks to this open-source character, very significant structural changes could be made in order to build a platform with independent modules that are easily manageable and adaptable to the new needs of the project. In addition, the purchase price is cheaper than other proprietary solutions and all plans and features are available to the entire community free of charge, which is both an economic and knowledge advantage. Finally, the vehicle has a transition system from manual to autonomous driving, so that both driving modes can be used on the platform.

2.2.2 Sensors

The vehicle has a roof rack on which the three main sensors currently installed are mounted: LiDAR, RGB camera and GPS. This structure (shown in Figure 2.3) is made of aluminium profiles and allows for easy handling and adaptability of the arrangement of the sensors. This quality makes the system very versatile. These sensors are connected to the on-board computer and send the information obtained in real time for further processing.

- **Velodyne VLP-16 LiDAR**: a 3D LiDAR consists of stacked rotary lasers that allows a 3D reconstruction of the environment through a point cloud following the same mode of operation as an ultrasound sensor. VLP-16 LiDAR is the smallest and one of the most advanced in VLP series. It has 16 channels and a vertical field of view of 30 deg. (arranged as $\pm$ 15 deg.). It can detect targets in a range of 100 m. and its rotation rate is between 5-20 Hz.

- **ZED Stereo Camera**: Computer vision has proven to be a great ally in object detection and tracking, as well as a valuable input for Deep Learning based solutions. In this case, a stereo camera with two lenses is used, mimicking human vision. This binocular approach allows for an in-depth representation of the scene to achieve a 3D visualisation. With a depth range of up to 20 metres, the Zed camera can be used in conjunction with technologies such as ROS, OpenCV, CARLA or MATLAB. Together with LiDAR, they both form the diarchy of sensors that govern the Perception Layer of this proj ect.

- **Topcon Hiper Pro GPS**: As an indispensable sensor for the Localization Layer, RTK-GDPS is presented. It is a system that makes it possible to obtain the position at any point on Earth with centimetre precision. It is able to operate based on triangulation obtained from a network of satellites that are deployed in the orbit of the planet.

It should be noted that a complete sensory architecture has a greater number and diversity of sensors, because the greater the number of sensors, the greater the understanding of the environment. This concept is known in autonomous driving jargon as sensor redundancy. In Chapter 4, the functioning and operation of LiDAR and RADAR will be discussed in more depth, and the other sensors typical of an autonomous driving architecture will be introduced.

2.3 Localization Layer

Localization Layer is in charge of positioning and locating the vehicle on a map with a real-time and centimetric accuracy approach. As previously mentioned, this operation is empowered by Topcon Hiper Pro GPS which depends on a antenna that is deployed at Escuela Politécnica Superior and acts as local base station.

Currently, *Techs4AgeCar* architecture uses a multi-constellation system with a Real-Time Kinematic differential positioning solution [4]. There is an additional odometry system based on encoders which allows position estimation, measuring the movement of the rear wheels.

Then, the information of both systems is fused in a Extended Kalman Filter [5] for better accuracy.

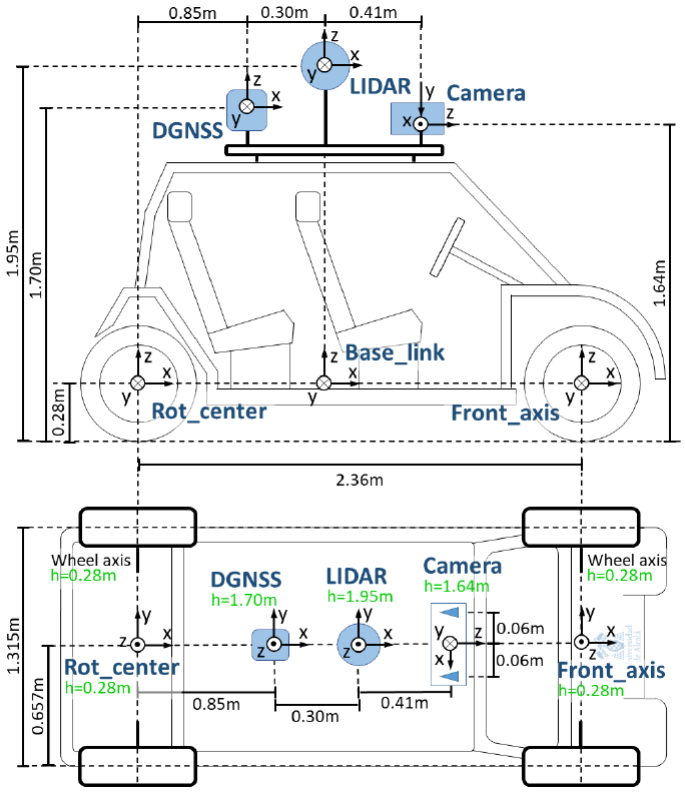


Figure 2.3: Arrangement of sensors at *Techs4AgeCar* autonomous vehicle
Source. Drive-By-Wire Development Process Based on ROS for an Autonomous Electric Vehicle [3]

2.4 Control Layer

Control Layer is responsible for guiding the vehicle when operating in autonomous mode. To do this, it generates the commands that are sent to the actuators. Currently, a Waypoint Tracking Controller has been implemented [6] (diagram shown in Figure 2.4). In order to know which command to send, it receives as input some waypoints from the calculations made in Planning Layer. Once the waypoints are received, spline interpolations are performed and a velocity profiler ensures a smooth and continuous trajectory. This allow it to reach its destination while avoiding obstacles and in optimal direction and speed conditions.

This method sends linear speed and curvature parameters to Drive-by-Wire Layer. As an additional feature, Control Layer should ensure that navigation is secure. Therefore, it has the ability to perform emergency stops or braking, independently of the route generated by Planning Layer.

As a novelty, control algorithms are being redesigned using Deep Reinforcement Learning techniques. In an analogous way on how the current controller is designed, these algorithms are also based on a waypoint system.

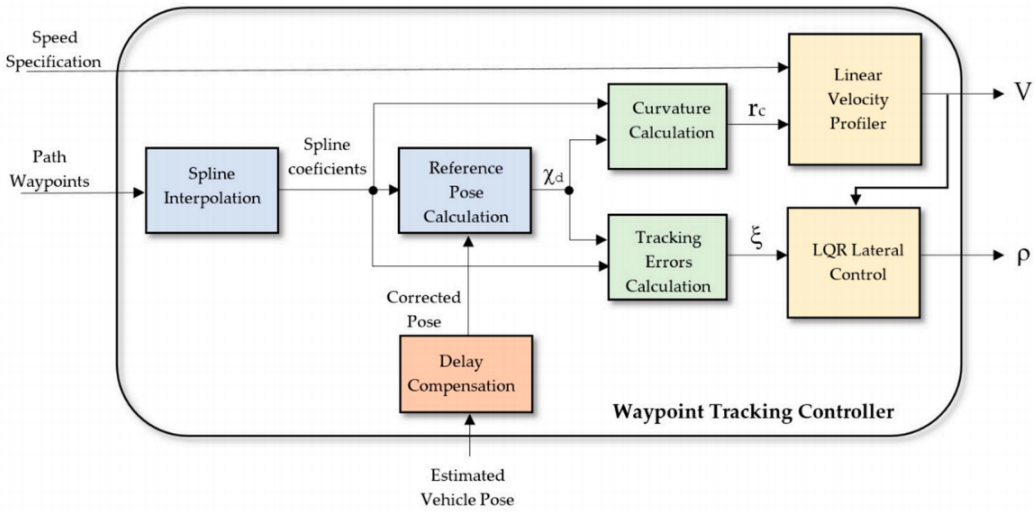


Figure 2.4: Waypoint Tracking Controller diagram
Source. A Waypoint Tracking Controller for Autonomous Road Vehicles Using ROS Framework [6]

2.5 Mapping and Planning Layer

This layer is responsible for the topological and geographical modelling of the environment through which the vehicle drives. For this purpose, it uses a HD Map approach which is supported by the OpenDRIVE software tool. For a more exhaustive and realistic analysis of the decision-making process, a custom map has been generated in CARLA Simulator that models the External Campus of the University of Alcalá, the place where the validations of the real prototype are carried out [7].

A HD map approach provides information about the road and the regulatory elements that appear on it, such as traffic signs, traffic lights or roundabouts. After receiving a route encoded as a point of origin and a point of destination, a route is generated using the A* algorithm, generating the waypoints used in Control Layer. The behaviour of this model has been evaluated in several scenarios such as Stop, Pedestrian crossing, Adaptive Cruise Control or Overtaking.

2.6 Decision-Making Layer

Decision-Making Layer is responsible for deciding what decision the vehicle should take taking into account the current circumstances of the vehicle and the environment, respecting the traffic laws. Currently, two different approaches coexist in the group. The first is based on Petri Nets programmed in C++. A Petri Net is a mathematical or graphical representation of a discrete event system in which the architecture of a parallel system can be described.

The second one is based on Partially Observable Markov Decision Process (POMDPS). It is a framework in which is possible to model a variety of real-world sequential decision processes such as robot navigation or autonomous driving. Both systems are fed with information that comes from Perception and Mapping Layers.

2.7 Perception Layer

To ensure safe autonomous driving, it is necessary to develop a safe and robust Perception Layer that is able to understand the environment around the vehicle thanks to the information collected by sensors. As mentioned above, in the current project architecture, the two sensors mainly used in this layer are RGB cameras and LiDAR. Therefore, we can tell the reader that one of the objectives of this TFG is to explore the operation and functioning of RADAR and the advantages it can bring to the current system for its possible incorporation in the future.

Currently, the system developed by the group is SmartMOT, whose structure is shown in Figure 2.5. It is defined as a real-time and power-efficient Multi-Object Tracking (MOT) pipeline used to predict the future behaviour of multiple agents in the scene. Firstly, the two sensors in charge of environment perception detect static objects (road, trees, traffic signs) and dynamic objects (pedestrians, vehicles or cyclists) that can be classified as road participants or road context information. Objects are detected after a PointPillars-based algorithm [8], [9] and YOLO-based [10] processing and a further information fusion. Sensor fusion, HD Maps information and ego-vehicle status are the inputs that feed the tracking module.

Then, Bird Eye View Kalman Filter [11], [12] and Hungarian algorithm [13] are used to perform state estimation and data association, respectively. This allows the detection of the most relevant objects in the scene and the prediction of their future movements, information that will feed the vehicle's executive layer.

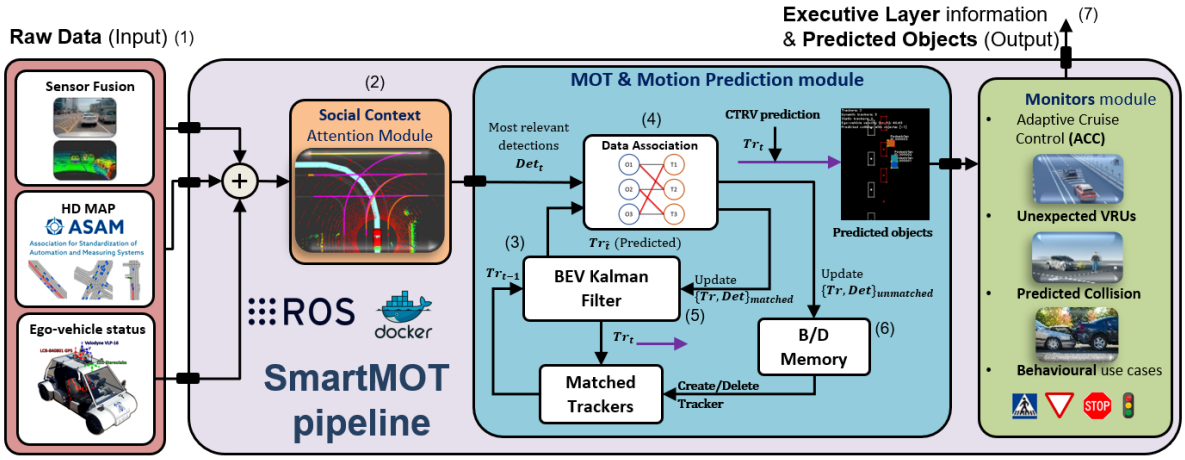


Figure 2.5: *SmartMOT* pipeline diagram
Source. RobeSafe Research Group

Historically, RobeSafe has included computer vision among its lines of research. This is why it has developed solutions for object detection and tracking using image processing and Deep Learning techniques. One of the great achievements of the group was the development of ErfNET [14], [15], a Convolutional Neural Network capable of performing semantic image segmentation efficiently and in real time.

2.8 Vehicle-to-Driver Layer

This module can also be referred to as Human-Machine Interface and is responsible for providing a bi-directional driver-vehicle interaction interface. It displays system status information to the user and the user can enter commands for the system to adopt a certain mode of operation. Currently, this HMI is implemented on two screens behind the steering wheel on the actual platform. The left display shows status information to the user: speed, battery, vehicle status information. The screen on the right is used for the user to decide which route the vehicle should take.

Another line of research related to this layer focuses on driver status and assessment. An architecture [16] is proposed to obtain visual attention maps based on passive sensors (cameras) mounted on the vehicle and tracking the driver's eyes. Its diagram is shown in Figure 2.6. For this purpose, the OpenFace 2.0 [17] framework is used.

The pipeline of this architecture starts with video acquisition from the ZED camera that focuses on the driver. This is followed by face detection and gaze estimation, taking into account that the camera must have been previously calibrated. After that, it is introduced into a visual focalization model that will return the desired attention map.

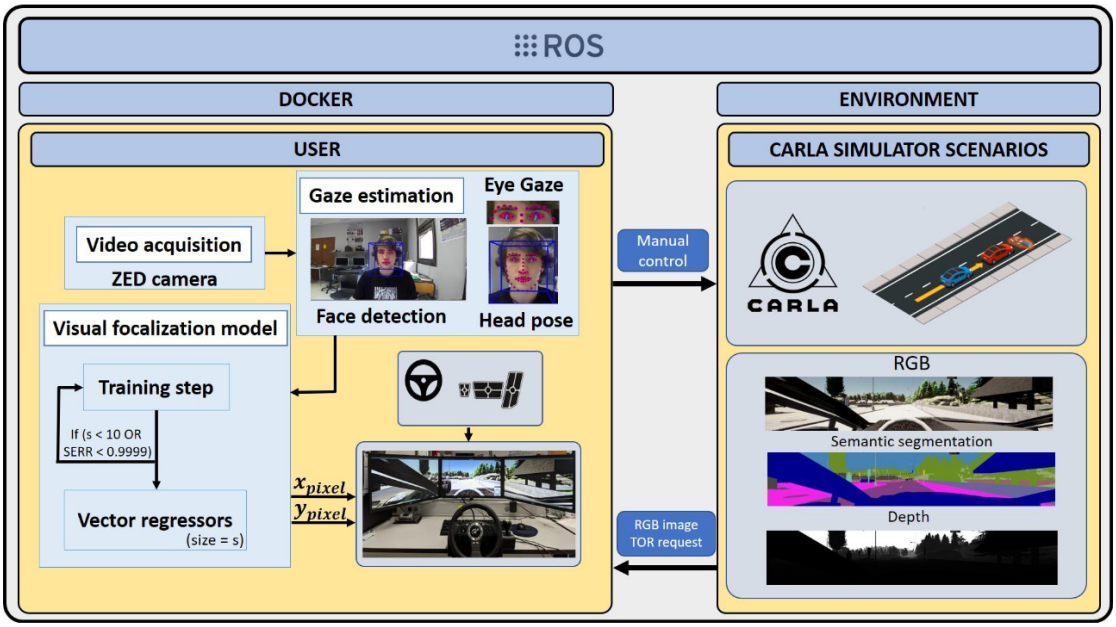


Figure 2.6: *Smart Vehicle to Driver* pipeline diagram
Source. RobeSafe Research Group

Chapter 3

State of the Art

Listen and you will be wise. The beginning of wisdom is silence.

Pythagoras of Samos

3.1 Introduction

One of the main objectives of this work is to contextualise the work in the Autonomous Driving topic, delving specifically into the Perception Layer. For this reason, this Chapter will review the literature and the technologies and architectures used at the frontier of the state of the art, even taking into account that the architecture proposed for this TFG will be presented in Chapter 6.

Perceiving the environment and extracting useful information in order to take appropriate decisions is a critical task for ADS [18]. Among the information that can be considered as interesting it is possible to find the free drivable areas and surrounding obstacles' locations, velocities, and even predictions of their future states [19]. This must be done in real-time to ensure a safe and reliable operation. The information extracted by the sensors is processed by Machine or Deep Learning models, which have established themselves as dominant and leading players in the state-of-the-art of this study field. Combining the strengths and the benefits of both, sensors and software techniques, the car can piece a detailed understanding of what's happening in the world.

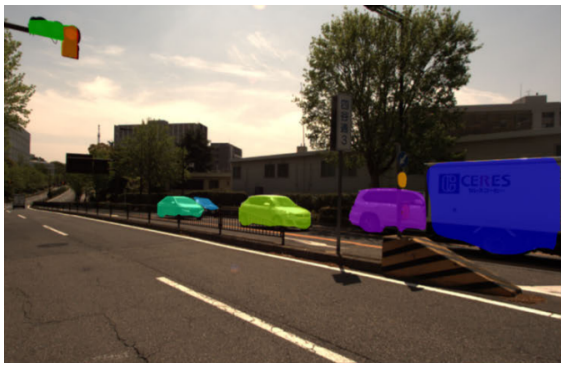
Perception Layer can be further subdivided into other categories: Object Detection, Object Tracking, Road Feature Detection, Sensor Fusion and Motion Prediction. During the following literature review, the two categories that fully involve the current project will be discussed in more detail: Object Detection and Tracking. On the other hand, the remaining categories will be slightly studied and presented.

3.2 Object Detection

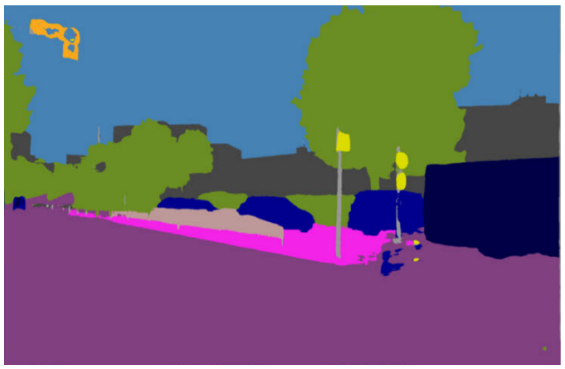
One of the most basic tasks to be performed in order to start recognising the environment is Object Detection. A task that may seem so trivial to a human being turns out to be key to the development of ADAS and allows vehicles to detect elements on the road in order to extract information with which to make decisions later on. Both static objects, such as traffic lights or semantic road information, as well as dynamic objects such as pedestrians, cyclists or vehicles are considered to be of vital interest in detection. During this Section, the detection of dynamic objects will be addressed, while in Section 3.3 more emphasis will be put on static objects that give context to the scene. Similarly, it can be defined as the set of techniques that allow extracting and locating instances of an object and its main characteristics based on data collected by sensors: images, videos or point clouds.

In Autonomous Driving applications, solutions based on Machine or Deep Learning techniques, which are supported by a wide variety of mathematical algorithms, have become the state of the art, the latter being the most robust and extrapolable. During the following Section, architectures of special interest to the reader and the author will be described, either because they were a breakthrough in the context of their technique and characteristics or because they revolutionised the paradigm of object detection by introducing new techniques.

(a) Bounding box results from YOLOv3 [20].



(b) Instance segmentation results from MaskRCNN [21].



(c) Semantic segmentation results from DeepLab v3 [22].

(d) 3D Lidar data results from SECOND [23].

Figure 3.1: Object detection techniques applied to an urban scene near Nagoya
Source: A Survey of Autonomous Driving: Common Practices and Emerging Technologies [18]

3.2.1 Image-based Object Detection and Semantic Segmentation

A. Image-based Object Detection

Although camera-based techniques and computer vision are not the direct focus of this work, a complete state-of-the-art study cannot overlook these techniques. And this decision is made because object detection has been a very traditional recurring task for computer vision scientists. For more than 50 years, techniques and algorithms have been proposed and developed for the extraction of features (corners, edges, silhouettes) that would result in the classification and detection of objects present in the image.

Object detection by computer vision became important in the context of autonomous driving in the late 1990s and early 2000s. It was not until 2012, when AlexNet [24] appeared, a convolutional neural network that managed to solve the ImageNet Image Recognition Challenge, that both industry and academia turned their sights and research trends towards Machine and Deep Learning algorithms.

In the state of the art, many of the methods rely on Deep Convolutional Neural Networks (DCNN). Furthermore, a clear distinction can be made between two subcategories of detectors: single stage detectors, which use a single CNN to detect the position and location of the object in space and also classify it into a class, and region proposal detection frameworks, which are architectures that differentiate two stages with two distinct networks, one used for proposing regions of interest and the other for refinement and classification.

On the one hand, single-stage detectors tend to have a small inference time and low memory cost, making them ideal for autonomous driving applications. One of the best-known single-stages detectors is YOLO [20], an acronym for You Only Look Once. The results of YOLO can be seen in Figure 3.1a, where it can be seen that the output of the model is a 2D bounding box that includes information about the detected object: its size and location, its class and the probability of belonging to that object class. The YOLO architecture is considered to be a great example of a single-stage detector, as it is composed of a single fully-connected DCNN that extracts the most important features from the image for further classification in the next layers of the network. This design makes YOLO very fast, reaching runtimes between 45-50 Hz for basic versions. Given the popularity of the architecture, other versions have appeared over the last few years, providing differentiating features between the different models, such as a minimum runtime in exchange for slightly less accuracy, or the other way around.

Another popular algorithm in the field is SSD [25], an acronym for Single Shot Detector. This model is based on standard DCNN architectures for image feature extraction, such as VGG [26]. With it, competitive results could be obtained in public benchmarks such as KITTI.

Other approaches include the use of multiple cameras to have a 360-degree view around the vehicle that allows it to reconstruct the entire environment of the vehicle, just as the point cloud of a 360-degree lidar would. Other parameters come into play in such solutions that have to do with the nature of the cameras and how their parameters are calibrated so that the camera pointing in a certain direction can communicate to the complete system that it has detected an object in a position that both camera and system understand.

B. Semantic Segmentation

On the other hand, computer vision scientists have not only been able to solve the problem of object classification and detection but have also gone a step further to be able to make object classifications in images at the pixel level. This task is called semantic segmentation. Therefore, each pixel of the analysed image will be associated with a semantic class such as car, vehicle, pedestrian or road. An example of semantic segmentation can be seen in Figure 3.1c from DeepLabv3 [22].

Going a step further, it is also possible to discern between different objects by analysing the pixels that are grouped within a class. Each object in the same class is called an instance and this task is called instance segmentation. Another example of this can be seen in Figure 3.1b from MaskRCNN [21]. In addition, these methods have been shown to have sufficient potential to become real-time solutions, which is necessary to be part of a detection system in the context of autonomous driving.

3.2.2 3D Object Detection

The main disadvantage of camera-based perception is that it takes place in two-dimensional space. Therefore, it is necessary to deduce the third dimension to obtain a complete spatial location of the detected object instance. Taking advantage of this, LiDAR and RADAR sensors appear on the scene to provide three-dimensional information natively. Eliminating this restriction by using sensors that offer 3D information allows for a more faithful reconstruction of the environment. For more information on data collection in 3D space, see Chapter 4.

This data is usually stored in 3D arrays and is often packed in point clouds. These data packets therefore encode the surfaces of the objects in the scene. In fact, sometimes the information and the number of points representing an object are small and scattered. Moreover, the greater the range or distance from the desired object, the fewer the number of detections picked up by the sensors. Faced with this complicated and adverse scenario for object detection and classification, Machine and Deep Learning algorithms were postulated as the dominant approach in academia, as was the case with image-based methods.

Traditional LiDAR object detection methods that rely on 3D information from point clouds use sequential pipelines in which the following stages are typically found:

- **Ground removal or filtering**: the first filtering step consists of removing the road plane from the point cloud. This can be done by algorithms such as RANSAC [27], which is an iterative method to estimate parameters of a mathematical model from a set of observed data that contains outliers. As it will be fit to the plane, inliers are part of the road while outliers are part of the objects of interest. Other approaches [28] make intelligent use of the cloud geometry to perform a first stage of filtering, providing a more user-friendly environment in which to apply the RANSAC algorithm consecutively over different regions of the plane.

- **Map-based filtering**: the second filtering step is possible if additional information is available. This could be done by using HD Maps or by fusing information from other sensors. If this semantic information of the environment is available and regions of points can be quickly associated with known elements, the detection process is speeded up.
- **Noise filtering**: the third filtering step prior to clustering the remaining regions is to remove the remaining noise from the point cloud. In order to consider that a region of points that has passed the first two stages is considered as noise, we can take into account criteria of size or location in space.
- **Clustering**: the ultimate goal of this pipeline is the detection of objects in the scene. Given that the point cloud has been filtered and the data has been cleaned to a large extent, Euclidean clustering algorithms or region-growing methods are used to shape and obtain a region of space that corresponds to a set of points that is grouped as an object of interest. Among the simplest and best known Machine Learning clustering algorithms applicable to the context of autonomous vehicles are the K-Means or Nearest Neighbours algorithm based on k-dimensional binary trees [29] or the DBSCAN [30] algorithm.

A similar approach to this one mentioned will be the one implemented and explained in Chapter 6. The use of various filtering techniques allows for a better understanding and classification of the environment surrounding the vehicle, as clustering algorithms can focus on the areas of real interest.

In the spearhead of the state of the art, a more exhaustive classification of the proposed methods for 3D object detection based on point clouds is carried out, thus separating the different academic proposals into different categories [31]: projection based, volumetric representation and point-nets.

A. Projection-based methods

As previously mentioned, the field of computer vision has been studying the problem of image classification and object detection for decades. In addition, a large amount of data stored in benchmarks is available for the evaluation of models developed for 2D detection.

For these reasons, models that project a 3D point cloud onto a 2D image by means of plane projections [32], cylindrical projections [33] or spherical projections [34] have been developed during the last years. In fact, since both 2D and 3D information of the scene is available, it is possible to obtain a 3D bounding box that wraps around the object of interest by undoing the projection made.

The limitations of this approach are that the projection of the 3D point cloud leads to a loss of information (depth is lost, but on the other hand it makes the computation faster) and the spatial information or location of the objects of interest is not explicit or native. As research gaps, the encoding of the input images to the models can be influenced. An improvement in these techniques, which so far are done by traditional vision techniques, could increase the goodness of these models. They could be replaced by learned methods.

Some of the most representative models proposed in this category are:

- **VeloFCN** [33]: this approach was published in 2016. Li et al. proposed a method in which a cylindrical projection mapping and a Fully Convolutional Network (FCN) were jointly used to predict 3D vehicles bounding boxes. The 2D projection of the point cloud will feed the input to the FCN, which will have two outputs. On the one hand, a series of layers and convolutions will result in a map that classifies whether each of the points in the image belongs to a vehicle or to the background. On the other hand, the second output of the network determines the vertices of the bounding box that will delimit each of the vehicles, conditioned by the results of the first output.

- **C-YOLO** [35]: Simon et al. proposed in 2019 a method that managed to meet one of the industry's greatest needs: to operate in real time in order to be part of safety-critical systems. This approach uses a YOLO-based single-shot detector extended for 3D bounding boxes extraction and orientation regression. The architecture achieved a 50 fps runtime, more than any previous method.

- **MV3D** [36]: Chen et al. proposed a method in which various LiDAR projections, supported by sensory fusion with cameras predicted oriented 3D bounding boxes in 2017. Mainly based on the bird's eye view detection approach, this method combines the information obtained in a frontal projection with a bird's eye view projection. Both are obtained from the same LiDAR point cloud and are processed by feature extractors to obtain feature maps through a fusion done by a second neural network. One of the main contributions of the publication is the proposal of a deep fusion architecture, which allows the neural network layers to combine information from all sensors through interactions between the layers.

- **BirdNET** [37]: Beltrán et al. proposed in 2018 a method in which a point cloud representation was normalized. This method allows a generalisation of the sensor parameters of different LiDAR sensors and calibrations, offering a more robust solution to the variation of these parameters. The framework consists of a three-stage pipeline: in the first stage, the 3D LiDAR data is projected in a bird's eye view with a coding that allows to keep parameters of the representation that were lost in previous methods. Then, a convolutional neural network specialised in image processing performs the detection of objects on the projected image. Finally, in the third stage, the 3D detections are obtained in a post-processing phase.

- **PIXOR** [38]: Yang et al. proposed in 2018 a real-time 3D object detection from point clouds based on Bird's Eye View (BEV) model. PIXOR is a single-stage detector whose output is estimates of the 3D objects of interest from a pixel-by-pixel decoding of an image from the aforementioned perspective. This image was processed by a neural network that analysed at the pixel level. Assuming that the detected objects are at ground level, PIXOR stands out for its effectiveness while being able to maintain a running rate of 10 fps. This speed is achieved thanks to single-stage detection, the main contribution of the architecture, since the academy standard was on multi-stage detectors.

B. Volumetric methods

In an attempt to generate regions of interest on which to apply methods in a systematic way, volume-based 3D detection methods appear. These architectures generate a three-dimensional representation of the world that is bounded by cubes (a 3D grid), which are called voxels. Fully Convolutional Networks (FCN) are then used to predict object detections.

In this approach, each voxel has properties and attributes such as binary occupancy or continous point density, which allow a characterisation of each region. One of the great advantages of this category of methods is that the information is explicitly defined. However, one of its major disadvantages is that empty regions (which are not of interest for the analysis) will be processed in the same way as the cells of interest. This results in low performance and efficiency in the computation of results. Furthermore, having to perform 3D convolutions during processing, given the nature of the data, also decreases efficiency and computational time.

- **3DFCN** [39]: Li et al. proposed in 2017 a method in which the problem of object detection was addressed using single-stage FCN on a entire volumetric representation of the scene. This model only detected vehicles and its input was a binary volumetric input: vehicle or not. One of the main contributions of this publication was the application of 3D convolutions to the field of point cloud processing. This was a significant improvement over its competitors of the year, 2D or projection-based detectors, but it significantly increased computational times and decreased efficiency. The model consisted of two outputs. On the one hand, the first output predicted the voxels of interest in which the objects were located, while the second output predicted their respective coordinates.

- **Vote3Deep** [40]: researchers set out to increase the speed and efficiency of this approach, which had been so successful. In 2017, Engelcke et al. presented a model that reduced the complexity of the 3DFCN architecture and replaced 3D convolutions with sparse convolutions. For this, the L1 norm and Rectified Linear Unit (ReLU) activation functions were used. The great improvement in efficiency came from the assumption that all bounding boxes detected by the model should have a fixed size. Another major improvement over its predecessor was that this architecture was able to classify its detections between car, pedestrian and cyclist.

- **SECOND** [23]: driven by the impetus to bring voxel-based 3D sensing to a computational speed similar to that of competitors in other approaches, Yan et al. proposed SECOND in 2018. The main contributions of the publication were focused on the improvement of sparse convolutions, which allowed them to be faster and achieve the desired goal of state-of-the-art computation times. In addition, new concepts for improved object orientation estimation and new methods for data augmentation were introduced for LiDAR-only solutions that increased speed and performance. Its qualitative results can be seen in Figure 3.1d. As can be seen in Table 3.1, the benchmarking results on the KITTI dataset were very satisfactory for the object class car.

- **Voxel R-CNN** [41]: this architecture was proposed in 2020 by Deng et al. It is worth mentioning that in 2020, the main competitors of voxel-based detectors were no longer

projection-based detectors but point-based detectors, which will be explained below. The main objective of the publication was to boost the efficiency of the voxel-based methods, assuming that there is no need for localisation as precise and accurate as that offered by point-based solutions. Taking the voxels as input to the system, the architecture generates dense regions proposals from the bird's eye view representation. After this, it uses voxel RoI pooling to extract region features from 3D voxel features for further refinement, which turns the pipeline into a two-stage detector. According to the authors, taking full advantage of the voxel representation allows them to strike a balance between accuracy and efficiency.

C. Point-Net methods

Point-based object detection methods take as input the raw information from point clouds, so the key concept for object detection is the iterative clustering and sampling of groups of points. When the concept was first developed, one of the main challenges was how to include point cloud-like data in a neural network. The points in a cloud are sparse and not regular in terms of size or shape. At each scan of the sensor, the number of detected targets can vary depending on the scene and the range of objects around it. This problem of taking the entire point cloud was handled using the point-net concept, where a compromise between information loss and the desired cloud size is achieved through intelligent quantization of the original cloud input.

Considering the entire point cloud may be considered a disadvantage, as the computation time may increase significantly with respect to the number of points. In addition, the variation may not be proportional, but may increase logarithmically or exponentially as the number of points to be studied increases. In Machine and Deep Learning, a concept called Big-O notation is defined that provides an algorithm with an estimate of the evolution in its computation time with respect to the number of inputs it receives.

The first architecture to exploit this concept was PointNet [42] in 2017 by Qi et al. The model took as input segmented 3D point clouds on which object classification tasks were performed by partial cloud segmentation. Empowered by a neural network working at point-wise level, transformations were performed using convolutional layers and max pooling layers. The model did not give importance to the input order of the elements, as neural networks trained to work with images do. The first results of this network improved the results of volumetric detection methods and quickly became the new trend in the industry. The work started in PointNet was continued in PointNet++ [43] in the same year. In this second version, new features such as hierarchical sorting were added. Deeper and more sophisticated features could be extracted by splitting the original network into smaller networks and applying the original PointNet to each new region in a hierarchical way.

- **VoxelNet** [44]: it is considered to be the first architecture to achieve very satisfactory results in the field of 3D bounding box detection by consuming the entire raw point cloud as input. Proposed by Zhou et al. in 2018, it drew on the influences of its volumetric predecessors, as it generated random voxels composed of cloud points from which it

extracted features using a Voxel Feature Extractor (VFE). After this phase, it was fed to a neural network that performs 3D convolutions and proposed 3D regions in which to predict the bounding boxes, as well as their size and class. The training for the recognition of each of the classes is performed with independent models and voxels of different sizes so as not to lose detail of the scene.

- **F-PointNet** [45]: proposed by Qi et al. in 2018, this method needs the help of a monocular camera to operate correctly. The proposed architecture first performs 2D bounding box detections based on the camera information. This information is extrapolated to the 3D world through the intrinsic and extrinsic parameters of the camera nature and its calibration, respectively. Already in the 3D world, the camera information acts as a region of interest for the subsequent LiDAR point cloud analysis. These regions are called frustums. A two-stage detection using PointNet is applied to each of the frustums to perform a classification and regression of the 3D bounding boxes of the objects.

- **IPOD** [46]: proposed by Yang et al. at the end of 2018, IPOD is a 3D object detection framework based on raw point cloud. Each of the points that make up the point cloud are taken as elements to which object proposals are given. The training of this model was performed with labeled groundtruth. Once the points have been analysed and with the help of context and local information, each of the proposed regions is subjected to a PointNet treatment. Unlike its predecessors, it constructed the regions of interest to be treated from the points themselves, not from voxels or bird's eye view projections of the cloud. This is why one of the main challenges faced by the model was the possibility of information redundancy given the huge number of points. To address this, a neural network that filtered the points using 2D semantic segmentation techniques was used to support the model. As a result, the model obtained state-of-the-art results on the KITTI dataset, highlighting a high recall score (a metric that will be explained later in Subsection 3.2.3).

- **PointPillars** [8]: proposed by Lang et al. in 2019 (Figure 3.2), appears willing to continue research along the lines of PointNet, as by this year it had already established itself as a dominant trend in the state of the art. He acknowledges that the 3D convolutions or voxel splitting of his predecessors continue to be a bottleneck to achieving a robust 3D detection model in real time. Therefore, they propose a model that uses only 2D convolutions with a novel encoder that learns scene features in pillars (vertical columns). Thanks to the pillars, key operations are performed on a two-dimensional plane that is efficiently optimised for GPUs. In other words, the pillars allow the 3D scene to be converted into a 2D pseudo-image. Moreover, the model does not need hand-tuning to adapt to new scene or sensor conditions, and the authors claim that it is even valid for use with radar point clouds. It should be noted that the output of the model is the predicted 3D bounding boxes of the objects.

- **PointRCNN** [47]: proposed by Shi et al. in 2019. It consist on a two-stage 3D object detection framework. In the first stage, a bottom-up model is proposed in which 3D bounding

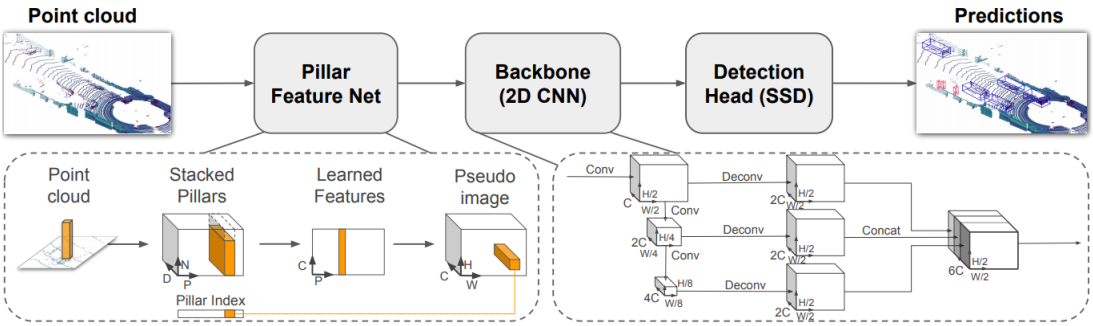


Figure 3.2: Pointpillars Network Overview
Source. PointPillars: Fast Encoders for Object Detection from Point Clouds [8]

box proposals are generated. Unlike its predecessors, the proposals are generated by means of an encoder-decoder that takes as input the scene information in point cloud format, which is then processed by means of a point-wise feature vector. This results in a series of 3D bounding box candidates, after a segmentation of the points that make up the foreground. The second stage is called Canonical 3D Box Refinenemt and is a model that is fed with all the intermediate results of the first stage. With the spatial information of the original point cloud, the semantic features extracted in the point-wise feature vector, the foreground mask and the 3D regions of interest that compose the candidates, a processing is performed that results in 3D bounding boxes of the objects detected with a high score in the evaluations in the KITTI benchmark, which outperforms their direct competitors.

- **SE-SSD** [48]: proposed by Zheng et al. in 2021, Self-Ensembling Single-Stage Object Detector is a model based on teacher-student architecture in which both network are single-stage detector. Both networks work together to obtain the predictions of the 3D bounding boxes of the detected objects. The targets detected by the teacher are called soft targets, while those detected by the student are hard targets. With both detections, an effective IOU-based matching strategy is designed to filter the soft targets and align them with the student's hard predictions by formulating a consistency loss. Currently, SE-SSD has one of the best marks compared to previous publications in terms of Average Precision and inference time. In addition, it is the top 1 in detections for the car classes for the 3D moderate and BEV moderate detection categories in the KITTI benchmark.

D. RADAR-based methods

Similarly, RADAR data offers a 3D representation of the world and are also stored in point clouds for further processing. The RADAR point cloud is not as rich in number of points and not as precise as LiDAR's but it is capable of natively obtaining the velocity of objects over a long range of distance. However, this low accuracy is a major disadvantage when used for Autonomous Driving purposes.

Algorithm	Time [s]	Easy	Moderate	Hard
PointRCNN [47]	0.10	85.9	75.8	68.3
PointPillars [8]	0.02	79.1	75.0	68.3
SECOND [23]	0.04	83.1	73.7	66.2
IPOD [46]	0.20	82.1	72.6	66.3
F-PointNet [45]	0.17	81.2	70.4	62.2
VoxelNet (Lidar) [44]	0.23	77.5	65.1	57.7
MV3D (Lidar) [36]	0.24	66.8	52.8	51.3

Table 3.1: Average Precision (AP) in % on the KITTI 3D Object Detection test for class car
Source: A Survey of Autonomous Driving: Common Practices and Emerging Technologies [18]

Another major disadvantage is its limited field of view, which can be overcome by the inclusion of several RADARs, an option made possible by its low price (and very low compared to LiDAR). These limitations have not prevented RADAR from playing a leading role in the development of ADAS components. The use of RADAR is widely expanded for proximity warning systems or adaptive cruise control systems.

However, thanks to their advantages and strengths, it is very common to see full autonomous driving systems in which LiDAR and RADAR work as a tandem, bringing the best of each to obtain as much high-quality information as possible. RADAR contributes its very long range, low cost and robustness to adverse climate scenarios while LiDAR offer precise object localization capabilities.

3.2.3 Evaluation metrics for Object Detection

To assess the goodness of a detection, it is necessary to apply industry standard metrics. These are usually based on a Ground-Truth, a term that refers to the knowledge of the solution of the problem by providing an ideal expected solution. This means that the data is required to have been previously labelled. Basic concepts must be introduced before explaining detection metrics, because a detection can be considered as:

- **True Positive (TP)**: correct detection done by the model. Point included in the ground-truth.
- **False Positive (FP)**: wrong detection done by the model. The model considered that the evaluated point or pixel was a target but it was not included in the ground-trouth.
- **True Negative (TN)**: both, model and ground-truth, considered the point as not-target. Not common term.
- **False Negative (FN)**: a point included in the ground-truth not detected by the model.

Now it is time to explore some of the most important evaluation metrics in Object Detection, that are:

- **Precision**: ability of a model to identify only the relevant objects. It is the percentage of correct positive predictions.

$$Precision = \frac{TP}{TP + FP} \tag{3.1}$$

- **Recall**: ability of a model to find all the relevant cases. It is the percentage of true positive detected among all relevant ground-truths:

$$Recall = \frac{TP}{TP + FN} \tag{3.2}$$

- **Precision-recall curve**: a graph in which Precision values are on the X-axis and Recall values are on the Y-axis. Both variables have a range $[0, 1]$. It can give a visual representation of both parameter and the greater the area under curve, the better.

- **Intersection over Union (IOU)**: or Jaccard index, is a coefficient that measures the degree of similarity of two mathematical ensembles and it consists on the division of the intersection of both ensembles divided by its union. In Self-Driving Cars jargon, we can measure this index either in 2D for camera or projected object detection or in 3D if we are working with point clouds. Its range is $[0, 1]$ and the greater, the better.

$$IOU(A, B) = \frac{A \cup B}{A \cap B} \tag{3.3}$$

- **Average precision (AP)**: the area under curve of PR curve is also known as AP. This metric appeared as a numerical value that facilitate the comparison between different curves, as it is not a trivial task to perform visually.

$$AP = \sum_{n} (R_n - R_{n-1}) \cdot P_n \tag{3.4}$$

- **Mean average precision (mAP)**: it is common practice to evaluate object detection models with different IOU thresholds, where each threshold returns a different score. In addition, there may be several classes (Q) to evaluate in the model such as: pedestrian, car, traffic sign. To solve these two unknowns and give a global measure of the model for several classes, this metric appears, defined as:

$$mAP = \frac{1}{n} \sum_{Q=1}^{Q=n} AP_Q \tag{3.5}$$

3.3 Road and Lane Detection

The issue of detection of dynamic objects coexisting in circulation with the ego-vehicle has already been extensively discussed. Now it is time to pay attention to the rest of the elements that make up the track in a static way, such as the ones seen in Figure 3.3. These elements will allow a complete understanding of the semantics of the scene and will allow us to make decisions on how to drive on the road according to the current traffic legislation or the nature of the road, detecting elements such as intersections or roundabouts, while respecting road safety regulations.

In Section 3.2.1, in which we talked about semantic segmentation as an image processing technique, we were able to check one of the techniques used for road understanding, since among the categories or classes into which the networks can segment the image are adjustable surfaces or traffic signs. Thanks to subsequent image processing techniques and individualised to each instance, it would be possible to discern which are the traffic signs or which parts of the free surface are trafficable because they are part of the road or not because they are reserved for pedestrian and/or cyclist traffic.

(a) Traffic lights

(b) Traffic signals

(c) Road lanes

(d) Pedestrian crossing with users

Figure 3.3: Key elements of the road.
Source: Google Images

But while semantic segmentation has been advanced as a representative state-of-the-art method, other more traditional techniques are already present in various ADAS to aid and assist the driver while driving. Other simpler image processing techniques allow distinguishing the lane in which the vehicle is travelling. Lane Departure Warning (LDW), Lane Keeping or Adaptive Cruise Control (ACC) systems are thus designed. The latter are usually supported by a radar sensor that allows the speed of the vehicle to be inferred in front of the ego-vehicle

natively by using the Doppler effect. This system can be seen in vehicles production vehicles such as the Toyota Prius, which is considered level 1. Monocular cameras are not the only sensor involved in road feature detection tasks, the lidar sensor performs very well in plane segmentation tasks from its point cloud information. They perform this task by fitting geometric models or splines to the plane. Models that assume that the road lines are parallel or that integrate topological information such as lane splitting and merging have also been used.

One of the biggest challenges facing models tackling this problem is the diversity of the appearance of roads. Depending on the country and its legislation, road lines can be painted in white or yellow, with different widths or with similar meanings. There may be variants, circular reflectors or cat's-eyes, different colours to the typical ones and others. The other two major problems are shared with computer vision-based object detection: low visibility conditions and image clarity problems.

3.4 Sensor Fusion

One of the ideas that has been introduced during the previous Chapters has been that a perception system for a fully autonomous architecture based on information extracted by a single sensor is not conceivable. In fact, Tesla has seriously opted for a perception system based solely on cameras over the last few months and has raised controversy and discussion throughout the industry. The vast majority of the industry opts for sensing systems based on an array of sensors operating together. To obtain the best possible representation and understanding of the scene, it is necessary to combine the information from all sensors in one way or another. This is where the concept of sensor fusion comes in.

Sensor fusion is defined as the ability to jointly extract information from the inputs of several radars, lidars or cameras to obtain a model or image of the vehicle environment that is as realistic as possible. These models are more accurate because they combine the different advantages and strengths of each of them, as well as counteracting the disadvantages of the others. For example, the camera's dependence on weather or light conditions can be compensated for by lidar or radar operation. However, the point cloud information from these two sensors can be coloured with the information coming from the camera to obtain a 3D colour representation of the scene. The set of synergies or interactions that all sensors in the vehicle may have is often referred to as sensor redundancy. In the words of the Aptiv company blog: "In the realm of active safety, every radar, every camera, every lidar (every sensor in a vehicle) doesn't just add to the safety capabilities; it multiplies them."

Among the architectures or models proposed by researchers (Figure 3.4), it is very common to find the combination of LiDAR + camera. However, in industry and in current production vehicles, it is easier to find ADAS systems that are fed with information from RADAR + camera or with the fusion between the three sensors: LiDAR, RADAR and cameras. Other more exotic or less explored combinations is the fusion between LiDAR + RADAR (fusion on which this work focuses).

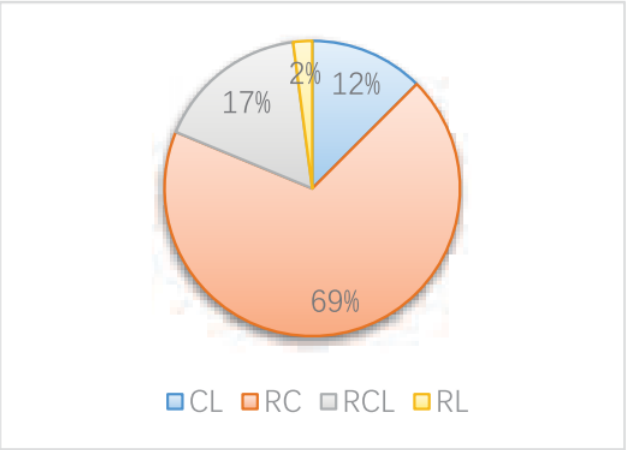


Figure 3.4: Status of multi-sensor combing form
Source. Multi-Sensor Fusion in Automated Driving: A Survey [49]

3.4.1 Motivation and benefits

Some of the improvements and benefits that come from the use of sensory fusion for perceptual systems have been mentioned during the introduction of the Section. Therefore, if the limitations of using a single sensor are as follows:

- **Sensor deprivation**: if a sensor breaks down or acts erratically in adverse scenarios, the perception system is left without reliable sources of information to act on.
- **Limited spatial and temporal coverage**: a single sensor has a limited field of view, which limits its ability to detect and track objects. In the field of autonomous driving, this limitation could be overcome with a single sensor, as 360-degree lidar is available. However, a full perception system would need more additional sensors to reach where this sensor does not.
- **Imprecision**: the measurements of a sensor are limited to its accuracy.
- **Uncertainty**: the uncertainty depends on both the nature of the sensor and the object detected. Both factors have an influence and it is more likely that key attributes or characteristics of the target will be lost for the understanding of the environment.

The use of multiple sensors will counteract this and the following advantages can be expected from their use.

- **Robustness and reliability**: the use of several sensors leads to the aforementioned redundancy. This allows to have a minimum of guaranteed information even in failure scenarios or bad conditions.
- **Extended spatial and temporal coverage**: if the sensors are oriented to cover a larger field of view as a whole, we will be able to detect and track targets for a longer period of time. If it leaves the field of view of one sensor, it will enter the field of view of another.

- **Increased confidence**: the confidence in the measurement of an object will be increased if the object is being detected by several sensors whose fields of view overlap.
- **Reduced ambiguity and uncertainty**: as a consequence of the previous argument, the fact that an object has greater confidence and more sources of data will reduce the uncertainties and ambiguities of the measure.

3.4.2 Types of Sensor Fusion

Sensor data fusion can be performed at different stages of the processing pipeline, or even before the processing pipeline starts. In addition, the information may be used to complement the disadvantages of another sensor or to increase the field of view and reach what the other sensor does not see. Given such casuistry and the range of opportunities offered by sensor fusion, a classification [50] is necessary to understand all the scenarios that can be solved by using this technique.

Three-Level Categorization

- **Early fusion or low-level**: the main objective of this type of fusion is to generate a representation of the environment by combining the raw information from the sensors involved. The result is in a way that would be impossible if only a single sensor were available. For example, there are models that use lidar and camera and from the information they both produce, generate a combination. There are architectures in which the lidar point cloud is coloured with information from the camera, and there are others in which each region of the lidar point cloud is assigned a probability based on what is sensed by the camera. Sensors of the same type can also be combined. Also, there are architectures in which 6 or 8 cameras pointing at different angles around the vehicle are used to obtain a 360-degree view of the scene using only cameras. In conclusion, such novel representations would be impossible to obtain using a single sensor.
- **Deep fusion or intermediate-level**: this fusion is the most complex and abstract of all, since it makes use of the architecture of the implemented neural networks themselves to fuse the information from the sensors and interrelate them in the different layers of the neural network in a hierarchical manner. This allows the features extracted from each input to be linked. That is, if we remember how a CNN works, it can be stated that each of the intermediate layers will be focused on the extraction of certain features: edges, corners, colours, etc. This is where the merging takes place, resulting in a general fusion scheme.
- **Late fusion or high-level**: in this type of fusion, the information from each of the sensors is processed in separate pipelines and it is not until the last stage of processing that fusion occurs. Normally, this occurs when some sensors are more capable of operating under the conditions of the scenarios (either due to the algorithm implemented or the complexity

of the scene) than the others, which act in a complementary manner. A clear example of this is the traditional combination of lidar and radar. Both sensors obtain point clouds, but given the different nature of the two, they have their strengths and weaknesses. The lidar point cloud has more points and detections than radar and therefore allows better detection of objects. However, the radar point cloud can complement its information, as it can obtain the speed of the targets and works better at long distances where the range of the lidar is not sufficient. This type of fusion will be discussed in more detail in Chapters 4 and 6, as it is the one that will be carried out during this work.

Configuration-based Categorization

- **Complementary**: several sensors are said to work in a complementary manner when all sensors are independent of each other and their information is combined to obtain a more accurate representation of the environment. Conceptually, it may resemble early fusion, but it is important to note that complementary sensors can be used to perform early fusion, if their raw information is combined, or deep fusion, if their features are combined within a neural network.

- **Competitive**: multiple sensors can be said to be competitively configured if each sensor is responsible for independent measurements of the same object. This configuration is used to create robust and fault-tolerant systems for each sensor. To differentiate them from the late fusion, we can exemplify this category by imagining an architecture that uses several radar sensors whose fields of view overlap. Why would we want this overlap? If several sensors detect the same object, we can combine all their characteristics and attributes obtained depending on the parameters of each of the sensors, and if both sensors give us the same detection with similar parameters, we can conclude that such a target strongly exists. Otherwise, we can discard false positives if one of the sensors has detected a target of interest and the rest of the sensors have concluded that it is not, because it is outside the conductive surface or other reasons.

- **Cooperative**: an array of sensors work cooperatively when information from them is used to generate raw information that could not be generated by one sensor alone. For example, a stereo camera generates a depth map by fusing the images from two or more monocular cameras. In contrast to competitive fusion, it usually decreases accuracy and reliability. A diagram that clarifies this classification can be seen in Figure 3.5.

3.4.3 Outstanding architectures based on Sensor Fusion

Although architectures that make use of sensor fusion have already been discussed when we talked about 3D Object Detection in Section 3.2.2, such as [36] [45], it is of interest to delve into more state-of-the-art architectures to see how scientists exploit the advantages of sensor combination to create the models that top the benchmark rankings.

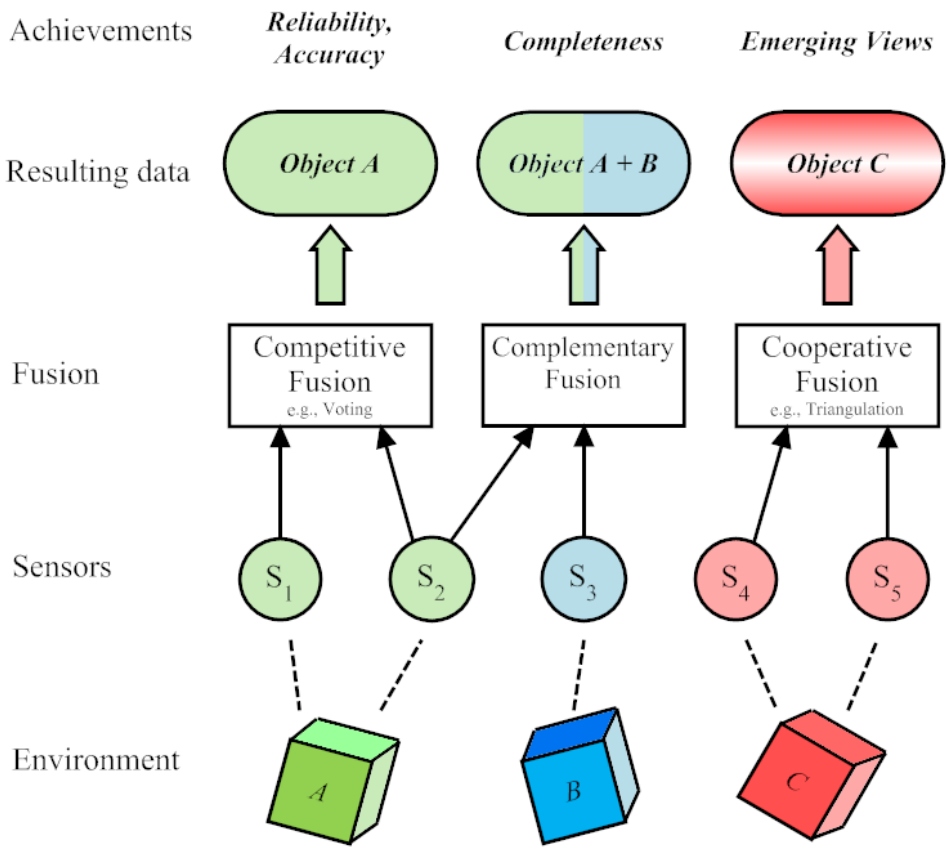


Figure 3.5: Complementary, competitive and cooperative sensor fusion Source. An Introduction to Sensor Fusion [50]

- **CenterNet v2** [51]: proposed by Yin et al. in 2021 at University of Austin. Powered by lidar, radar and camera information, it is a model that is ranked top 4 in the detection category for the nuScenes benchmark ranking. The model is capable of performing detection and tracking tasks thanks to its design. The key concept on which CenterPoint is based is that its predecessors found it difficult to orient objects and their 3D bounding boxes in space. Therefore, points are taken as 3D elements with orientation and velocity and are detected by a regressor in a first processing stage. In the second stage, a refinement process is carried out, adding more features to the detections. The result allows for greedy closest-point matching tracking.

- **PointPainting** [52]: in 2020, Vora et al. proposed another method to exploit the strengths of combining lidar and camera while improving the concept of PointPillars [8], whose architecture can be seen in Figure 3.6. This architecture is proposed because the authors found that during their year, lidar-only methods outperform fusion-based methods. This is blamed on a gap in the fusion literature. Therefore, to advance in this field, they propose to project the point cloud obtained with lidar using a PointPillars-based algorithm onto an image that has undergone semantic segmentation processing. In addition, point clouds from other lidar-only algorithms, such as PointRCNN or VoxelNet, are projected onto the segmented image. All of them show major improvements over their simple, single-sensor version.

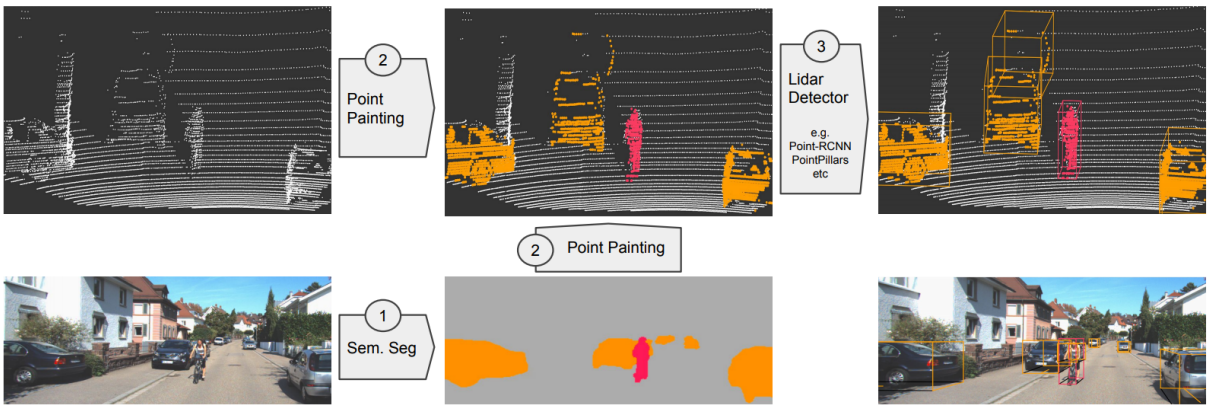


Figure 3.6: *PointPainting* Pipeline Overview
Source. PointPainting: Sequential Fusion for 3D Object Detection [52]

- **CLOCs** [53]: Pang et al. proposed in 2020 another method that exploited again the combined strengths of lidar and camera. CLOCs consists of a fusion architecture, as it combines individual 2D (camera) and 3D (lidar) detections as a consistent set of candidate hard detections. Both types of detections are normalised through the use of sparse tensors. Then, a 2D CCN is used to process these tensors, which are subsequently mapped to the initial inputs. From this processing, each of the initial candidates is then assigned a probability and score. The output of the system are those candidates (merge without lidar and camera detecting the same object) with the highest scores. An example of the output of the system can be seen in Figure 3.7.

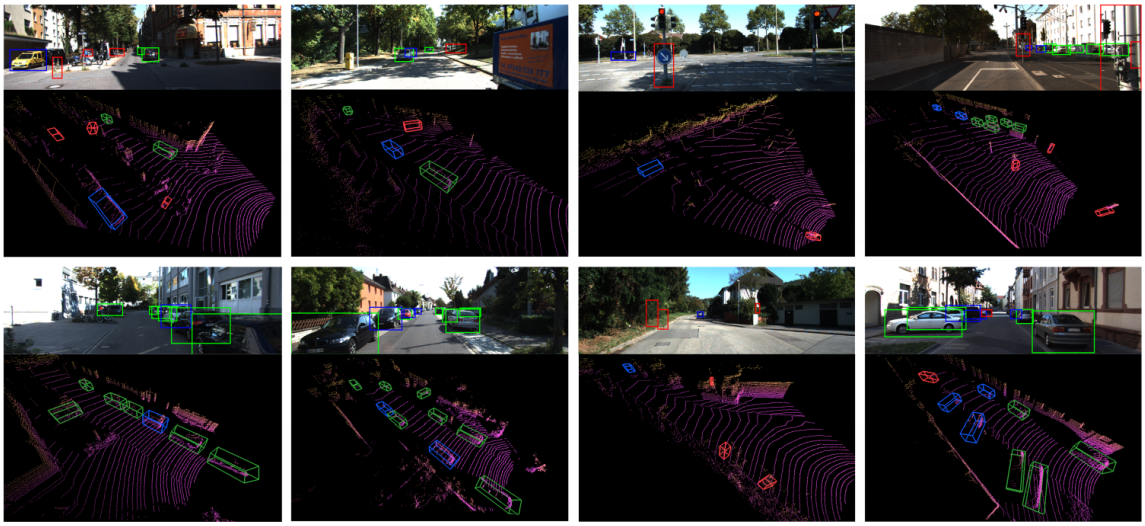


Figure 3.7: Example of qualitative results of a sensor fusion architecture
Source. CLOCs: Camera-LiDAR Object Candidates Fusion for 3D Object Detection [53]

As can be seen in Figure 3.7, the typical output of each of these systems can be seen as the representation of the point cloud in space, which in this case is coloured purple in the bottom row. While the 3D bounding boxes can be seen in red, blue and green. In the top row, their analogue 2D bounding boxes for camera detection can be seen.

3.5 Object Tracking

Object Tracking is a task that consists of estimating the state of an object present in the environment over time based on information from previous time instants. To achieve this goal, an initial data set must be taken and an identifier (ID) is created with which the object is tagged. Therefore, in the information for different time instants, instances of the same object must be found and labelled with the same ID as can be seen in Figure 3.8.

In contrast to Object Detection, traditional techniques have coexisted in recent years with Deep Learning-based techniques to perform this task. Although the latest architectures have opted for Deep Learning models for tracking or for architectures that mix detection and tracking within a neural network, traditional techniques have offered very good results that have solved the task in a very satisfactory manner. Among the most common traditional techniques we can find Kalman filters or particle filters or algorithms such as JPDA, which stands for Joint Probabilistic Data Association.

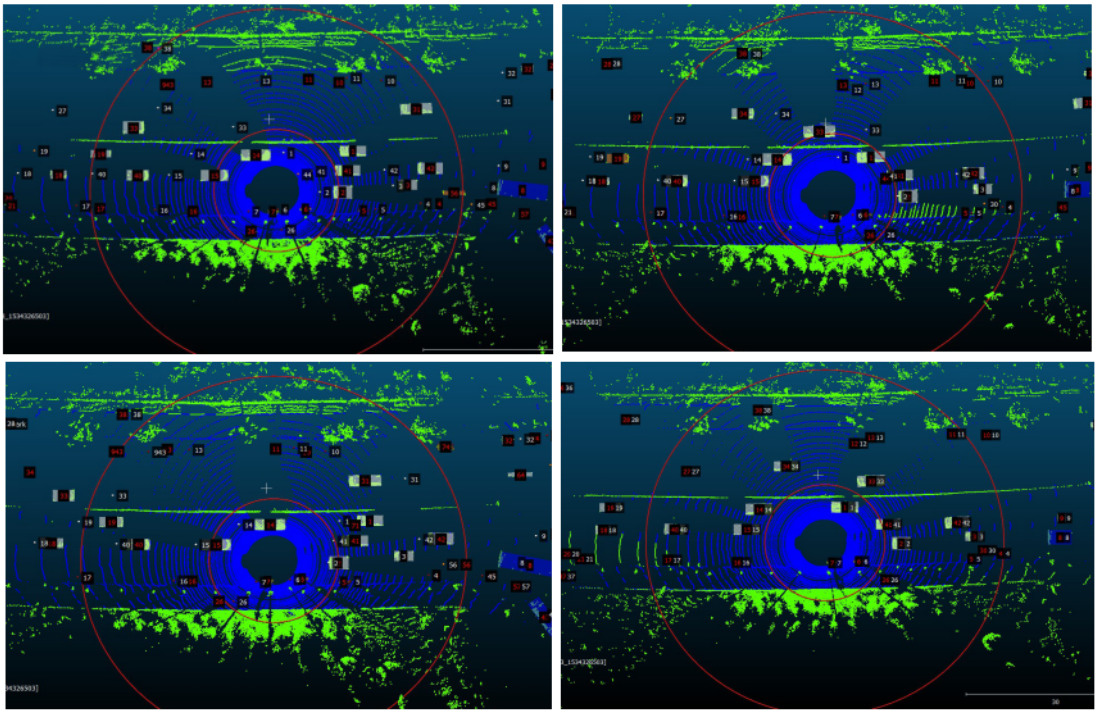


Figure 3.8: LiDAR 3D Object Tracking for several instants of time
Source. wad.ai. ICCV 2019 Workshop on Autonomous Driving

3.5.1 Fundamentals of Object Tracking

As we have seen throughout this chapter, there is a wide range of algorithms that fulfil the perception tasks in an autonomous driving system. It would not be otherwise in the case of Object Tracking. From this diversity of algorithms and approaches arises the need to establish a classification in order to discern which are the strengths of each one of them and for which application they are more suitable.

Detection-free vs. Detection-based trackers

In this dichotomy, trackers are classified according to how they are initialised and, consequently, how they will act when new objects enter the scene.

- **Detection-free Trackers**: in this type of tracker, the model must know the number of objects to be found in the scene and what they should look like. Therefore, this requires a prior initialisation done externally. In addition, all these objects are referenced in the first frame of the input to the system (point cloud or image). Therefore, the system will associate these objects during the following frames. It does not have a defined behaviour when other objects appear in the scene, as they are not considered to be of interest. Among the traditional techniques used in this method are pattern matching.
- **Detection-based Trackers**: in this case the model receives the objects to be tracked from a detector that has undergone processing prior to tracking. In the case of autonomous driving, this type of tracker is much more viable than the previous one, since it is robust and has defined behaviours when new objects appear in the scene. The context of an autonomous driving scene is more complex and cannot rely on external initialisation.

Single vs. Multiple Object trackers

This category serves to differentiate trackers depending on the number of objects they are capable of handling.

- **Single Object Trackers** (SOT): this type of tracker is able to track a single object in the scene even when multiple objects appear in the scene. This target is initialised in the first frame and tracked throughout the sequence.
- **Multiple Object Trackers** (MOT): these trackers are able to handle all relevant objects appearing in the scene. These MOTs are usually detector-based, so they are accompanied by a detector that makes them capable of tracking all objects and even those that are not initialised in the first frame of the sequence.

Online vs. offline trackers

In this case, algorithms or solutions should be differentiated according to their responsiveness or their ability to learn in the air or not.

- **Online Trackers**: such algorithms are able to continue to learn during runtime as well as provide real-time responses as they are deployed. These trackers usually draw bounding boxes in space and follow them.
- **Offline Trackers**: in this case, the tracker needs not be required to give a real-time response as it is not deployed.

3.5.2 Outstanding Object Tracking architectures

As mentioned, Multi-Object Tracking is an indispensable task for the development of autonomous driving systems. In such adverse conditions as the urban or road scene, being fully aware of the objects in it, not only of their position through detection but also of their temporal evolution, can be the difference between making a good decision or not. Analogous to what was done with Object Detection and Sensor Fusion, it is of interest to review some architectures proposed by scientists and researchers over the last few years in order to check the path towards which academia is heading and to further complete the goal of contextualising the problem of autonomous driving.

- **Tracking by Detection** [54]: Frossard and Urtasun proposed in 2018 a novel method to perform tracking by detection. This architecture is capable of exploit cameras and LIDAR data to produce accurate 3D trajectories. This system is composed of multiple neural networks that are interleaved in the architecture. Taking as inputs the RGB images from the camera and the lidar point cloud over time, a first stage proposes object detections. These detections are then redirected to the Matching and Scoring neural networks. After this, the problem is solved as a linear program on which an optimisation is performed on the trajectories or, in other words, on the position as a function of time. Concepts such as siamese neural networks or end-to-end learning are exploited in this architecture. The results were satisfactory on the KITTI dataset, where they outperformed their competitors on the MT and MOTP metrics.

- **AB3DMOT** [55]: the purpose of Weng et al. in 2020 for this publication was to establish a baseline of traditional techniques that provided similar results to the newer Deep Learning-based techniques. In addition, it proposed a set of guidelines for the development of 3DMOT systems based on this approach and a new family of metrics. A system capable of real-time operation is achieved by using the 3D Kalman Filter and a Hungarian algorithm to perform the tasks of state estimation and data association respectively. In addition to achieving results that reach state-of-the-art performance, this approach obtained the highest speed of all the entries recorded in the KITTI dataset, reaching an execution speed of 207.4 FPS. The new family of metrics offers a revision of the traditional metrics and offers new versions: average versions for MOTA and MOTP (AMOTA and AMOTP) and scaled average version for MOTA (sAMOTA).

- **GNN3DMOT** [56]: Xeng et al. also propose in 2020 a new approach in which they abandon the tracking by detection approach and opt for a feature extracting architecture based on Graph Neural Networks (GNN). This change of direction is taken to improve feature extraction and thus reduce confusion during the data association stage. The architecture consists of two separate feature extractors: a 2D and a 3D feature extractor, both extracting appearance and motion features using information from the current and previous time instants. Then, all the features of the objects are fused to build a graph. The model achieves state-of-the-art results with a runtime of 5.2 FPS.

- **TrackRCNN** [57]: Voigtlaender et al. propose in 2019 a new architecture that aims to extend the task of Multi-Object Tracking to a new concept of Multi-Object Tracking and Segmentation (MOTS). For the construction of the architecture, MaskRCNN was taken and extended to a 3D world. To do this, the original convolutions were replaced by 3D convolutions that include information about the temporal context. In addition, the head of the neural network was also replaced by one that produced association vectors for each of the detections to be tracked. The qualitative result of merging these two tasks can be seen in Figure 3.9. The quantitative results look promising and reach the state of the art in the tracking category. However, in the absence of datasets and benchmarks for this joint category, it remains to be seen whether this modality takes hold in academia.

Figure 3.9: Sample Images of Tracking and Segmentation
Source. MOTS: Multi-Object Tracking and Segmentation [57]

- **EagerMOT** [58]: in this approach, Kim et al. proposed in 2021 a new and simple method to benefit from sensory fusion for Multi-Object Tracking. The model consists of an effective multi-stage data association that can deal with detections from a wide variety of sensors, giving the system adaptive capabilities. The system can therefore be used for different tasks: 2DMOT, 3DMOT or MOTS and with different sensor configurations. The system takes as input the detections from a detector, either 2D or 3D. The detections are fused to get the instances of each object using parametric methods. The instances are then associated over time in two stages. The first stage contains information that is matched with existing tracks and in the second stage, all unmatched instances are matched with the instances, only in 2D space.

3.5.3 Evaluation metrics for Object Tracking

Building on the basis seen in Section 3.2.3, a similar process is carried out in Object Tracking as in Object Detection. In order to test and obtain quantitative results that prove the goodness of a model, evaluation metrics are used. These metrics became popular from [59], where Milan et al. proposed in 2016 an evaluation benchmark for Multi-Object Tracking architectures, called MOT16. Furthermore, in 2020, Luiten et al. proposed a new set of metrics in [60], with the intention of giving a higher-level view on the problem which enable better understanding of tracking behavior. Taking these two sources as a reference, a review will be made of the three main tracking metrics: MOTA, MOTP and HOTA.

- **MOTA** [59]: *Multiple Object Tracking Accuracy* is one the most used metrics for tracking evaluation. This can be understood because the way MOTA is defined makes it represent the three main sources of error in terms of tracking: false positives (FP), false negatives (FN) and identity switches (IDSW). The equation is defined for a frame t and GT is the number of objects present in the ground-truth.

$$\text{MOTA} = 1 - \frac{\sum_t (\text{FN}_t + \text{FP}_t + \text{IDSW}_t)}{\sum_t \text{GT}_t} \tag{3.6}$$

  Therefore, the right-hand side of the equation represents the total error made in the scene analysis. It should be noted that the MOTA index can become negative, although its range is intended to be defined from $(-\infty...1]$ or from or $(-\infty...100]$ if it is represented as a percentage (%). This negative situation occurs in the case where the number of errors during the analysis exceeds the number of objects present in the ground-truth.

- **MOTP** [59]: the *Multiple Object Tracking Precision* is a metric that gives a measure of the overlap between a detected bounding box and one that is part of the ground-truth of the scene. It is therefore a measure of the average similarity of True Positives and their ground-truth counterparts. It shows the tracker's ability to estimate positions accurately, so it can also be seen as a measure of position error.

$$\text{MOTP} = \frac{\sum_{t,i} d_{t,i}}{\sum_t c_t} \tag{3.7}$$

  In the equation c_t denotes the number of matches for a frame t, while $d_{t,i}$ is the bounding box overlap between the target i and its ground-truth counterpart. It can be noted that MOTP is an interpretation of accuracy in terms of spatial location.

- **HOTA** [60]: an acronym which stands for *Higher Order Tracking Accuracy*. It is a novel metric for evaluating MOT performance and designed to overcome many of the limitations of previous metrics. HOTA is a combination of three IoU scores that divides the tracking task into three sub-tasks: localization, detection and association. Each of the three subtasks will be associated with a pair of equations: the first one will evaluate and

define the task for a target and the second one will act by means of a Hungarian association algorithm to relate all the targets in the scene.

First sub-task is localization. Localization measures the spatial alignment between a predicted detection and a ground-truth detection. In other words, it is the ratio of the overlap between both mentioned detections by its union. Therefore, the expression defining this metric will be equal to the one mentioned in 3.2.3.

$$\mathrm{LocIOU}(A, B) = \frac{A \cup B}{A \cap B} \tag{3.8}$$

$$\mathrm{LocA} = \frac{1}{TP} \sum_{c \in TP} \mathrm{LocIOU}(c) \tag{3.9}$$

Afterwards, detection evaluation is carried out. Detection measures the alignment between the set of all predicted detections and the set of all ground-detections.

$$\mathrm{DetA} = \mathrm{DetIOU} = \frac{TP}{TP + FN + FP} \tag{3.10}$$

And the last sub-task is association. Association measures how well a tracker links detections over time into the same identities (IDs), given the ground-truth set of identity links in the ground-truth tracks.

$$\mathrm{AssIOU} = \frac{TPA}{TPA + FNA + FPA} \tag{3.11}$$

$$\mathrm{AssA} = \frac{1}{TP} \sum_{c \in TP} \mathrm{AssIOU}(c) \tag{3.12}$$

Finally, the three sub-tasks are linked in order to achieve a single metric to compare and rank the architectures that perform Object Tracking. Therefore, HOTA appears as a combination of the three IoU sub-metrics.

$$\mathrm{HOTA}_\alpha = \sqrt{\mathrm{DetA}_\alpha \cdot \mathrm{AssA}_\alpha} = \sqrt{\frac{\sum_{c \in TP} \mathrm{AssIOU}(c)}{TPA_\alpha + FNA_\alpha + FPA_\alpha}} \tag{3.13}$$

$$\mathrm{HOTA} = \int_{0 < \alpha \leqslant 1} \mathrm{HOTA}_\alpha \tag{3.14}$$

Chapter 4

Theoretical Study

We choose to go to the Moon in this decade and do the other things, not because they are easy, but because they are hard.

John F. Kennedy

4.1 Introduction

During this Chapter, an in-depth theoretical study will be made of those contents that form a direct part of the development of this work, although most of them have already been introduced in Chapter 3. Therefore, both Chapters pursue the objective of contextualising the reader in the problem of perception systems in the field of Autonomous Driving.

This Chapter consists on a review of the sensors that are currently used in the development of complete perception architectures. As the development of this work is focused on an Object Detection and Tracking system with LiDAR and RADAR, the physical fundamentals of both sensors will be reviewed in order to understand their key points and strengths in order to be able to propose a competent sensing architecture. This analysis will include physical concepts in terms of principles of operation and powers to be handled, as well as concepts related to the type of information they provide when representing the scene.

To conclude the Chapter, the advantages and disadvantages of both sensors will be put together to give an overview of what each sensor can contribute and what strengths in perception can result from the sensory fusion of the two. The analysis of both sensors and the fusion between them provides the basis for the architecture proposed in Chapter 6 and justifies the reasons for it.

4.2 Sensors Fundamentals for Perception in Autonomous Driving

As has already been repeated several times during the course of this work, a sensor suite that provides a complete perception of the environment surrounding the vehicle is composed of various sensors of different types that can work cooperatively, in coordination or in competition with each other.

In the field of autonomous vehicles, there is a triarchy of sensors that dominate over the rest to perform the tasks of the perception layer: cameras, LiDAR and RADAR, the latter being the most exotic of the three. In addition, other sensors provide additional information to obtain every last detail of the environment. Inertial Measurement Units (IMU), GPS/GNSS Location Systems, ultrasonic sensors or even sonar are sensors that are also normally included in an autonomous vehicle but either play a fundamental role in another layer of the vehicle or take on secondary roles in support of the three main sensors, e.g. short-range ultrasound for parking situations.

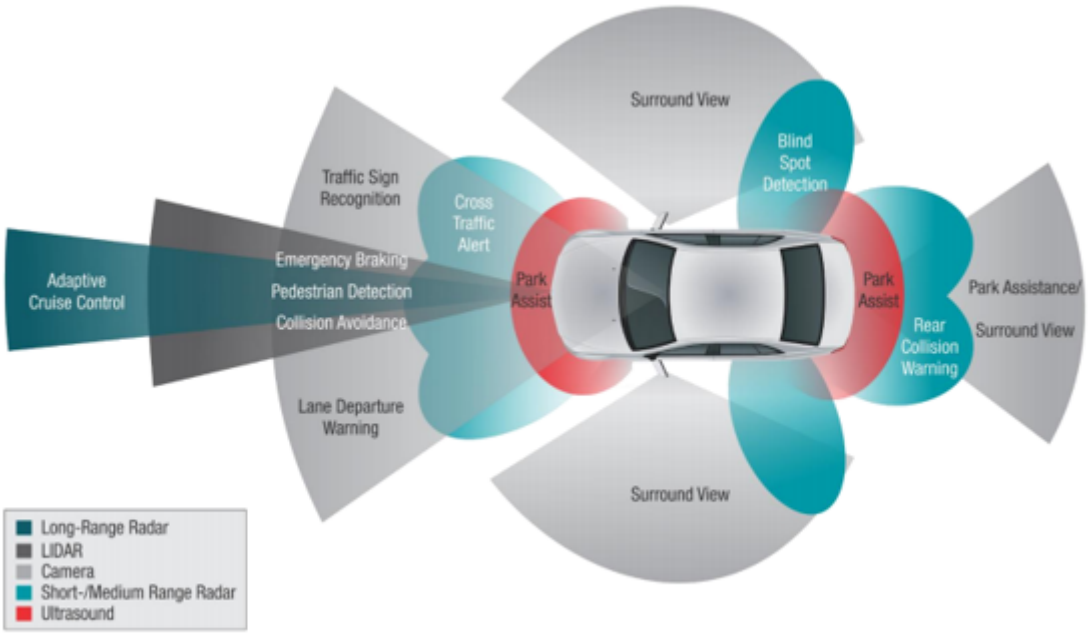


Figure 4.1: Typical disposition of Perception Sensors in a vehicle
Source. cdn.rohde-schwarz.com

A typical disposition of sensors can be seen in Figure 4.1. Furthermore, it is possible to observe how each of the sensors is best suited to perform different tasks due to its main characteristics and strengths. The combination of all of them results in the aforementioned concepts of sensor redundancy and sensor fusion that allow the construction of an increasingly robust architecture capable of adapting to and handling different scenarios.

During the next pages an in-depth explanation of the theoretical background of the sensors directly involved in the development of this work will be given: LiDAR and RADAR.

4.2.1 Fundamentals of LiDAR

LiDAR is an acronym for *Light Detection and Ranging*. It is a sensor that can be considered as the little brother of RADAR. It is a technology whose basic operation is based on the Time of Flight (TOF) concept. To measure the distance to a target object, infrared light or laser beams are emitted and the time it takes for the reflected beam to return to the sensor receiver is measured. Therefore, by establishing a relationship between the laser's round-trip time and the speed of light, this desired distance can be obtained. For these reasons, it is considered an optical remote sensing technology. As in this case we are dealing with light waves, their speed is $c = 3 \cdot 10^8 m/s$ and a target at 150 metres can be scanned in a time of $1 \mu s$. This makes it possible to collect thousands of points in a few milliseconds to create a 3D representation of the environment surrounding the vehicle.

Its first versions date back to the 1960s decade in which the laser makes its appearance on the scene. Its first major contribution to human history dates back to 1971, when LiDARs aboard the Apollo 11 space mission made a scanned map of the lunar surface. This feat brought out the sensor's mapping abilities. That is why long before it was intended for the automotive sector, it was included in other branches of knowledge, such as archaeology, geography or topography. In these fields, it is very common to equip aircraft with a series of LiDAR sensors to scan vast areas that are difficult for humans to explore and with a high density of vegetation that makes it impossible to know the terrain accurately with cameras alone. An example of such geographical areas where LiDAR can be of interest is the Amazon Forest. As already mentioned in Chapter 1, LiDAR has been of interest to the autonomous driving industry since the 2005 DARPA Challenge, where it proved to be of great help in digitally reconstructing the environment around the vehicle.

LiDAR is considered as an active sensor since it generates energy, in this case in the form of light, for its purpose. Sensors that only receive energy for their measurements are called passive sensors. The greatest exponent of passive sensors in the field of Autonomous Driving are cameras. The fact that LiDAR is an active sensor makes it robust to adverse scenarios, as it eliminates dependence on the environment. LiDAR performs well in low light or high brightness situations, situations in which the performance of a camera would be greatly reduced. However, there is one adverse scenario in which LiDAR does not perform well at all. Adverse weather conditions such as snow, rain or fog can be a problem for the performance of this sensor, as it does not perform at 100% of its potential under these circumstances.

This is because the principles of LiDAR operation can be explained by three simple concepts of optics and photonics: absorption, scattering and reflection of light. When navigating in adverse weather conditions such as snow, rain or heavy fog, the absorption of light by water and atmospheric scattering of the direction of light rays reduces the density of light capable of reflecting off an object and returning to the sensor to be treated as a detection. The fewer rays reflecting, the less likely that the object of interest will be detected. This is a disadvantage shared with the camera but inherent in the case of RADAR. Looking at the physical fundamentals of LiDAR, it could be understood as a hybrid between RADAR and camera.

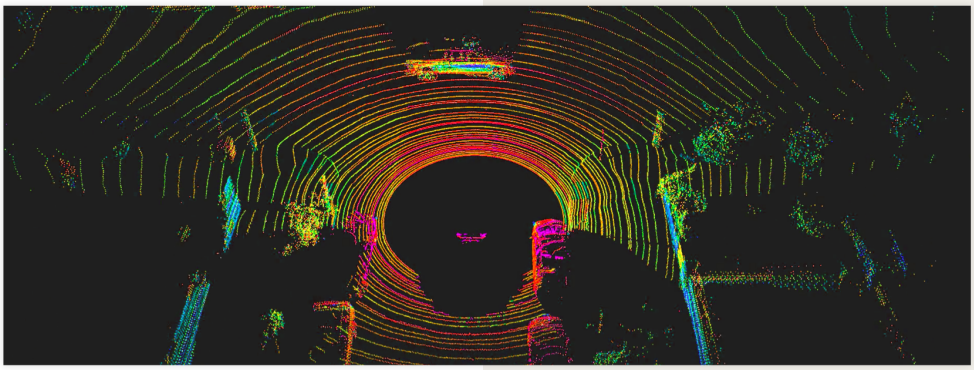


Figure 4.2: Typical representation of a 3D LiDAR point cloud
Source. voyage.auto. An Introduction to LIDAR: The Key Self-Driving Car Sensor

Listing some of the main features of LiDAR:

- **Range**: Both LiDAR and RADAR perform similarly in terms of vehicle detection range. Both can detect from close to a few metres up to 200 metres. However, it is true that one of the areas where it needs to improve is in the vicinity of the vehicle.
- **Spatial resolution**: One of the greatest qualities of LiDAR is its spatial resolution. The fact that it works with light beams (lasers) whose waveforms range from 905 to 1550 mm means that a resolution of 0.1 degrees is obtained. It is therefore possible, thanks to LiDAR, to distinguish objects that are very close in space or even have overlapping parts.
- **Field of view**: Both mechanical and solid-state LiDARs have an excellent horizontal FOV. In fact, by using a rotation mechanism, mechanical LiDARs are able to scan 360 degrees around the vehicle. As for the vertical FOV, it is arguably larger than that of its big brother, RADAR. In terms of angular resolution, both azimuth and elevation angles also have a high resolution. Consequently, the compendium of these advantages over other sensors in terms of resolution and FOV is the main reason why LiDAR has positioned itself as the most widely used sensor for object detection tasks.
- **Cost and size**: However, this is its major drawback. The early LiDARs, which were mechanically built, were very large and were mounted on the roof of the car. They were also very expensive, with a single sensor costing more than $50,000. The price alone meant that this technology could not reach the user, as it would increase the price of the vehicle by a factor of 2 to 3 times. With the advent of solid-state LiDAR and its increasing popularity, thanks to its good results, the industry has put its efforts into making the technology cheaper and reducing its size. Over the next decade, the price of this sensor is expected to be around $200, 100 times less than a few years ago.

After this short introduction to what LiDAR is and what are the main advantages it offers to the user, it is time to present the physical fundamentals and the main components inside one of these sensors.

LiDAR Power Equation

As LiDAR is a sensor that works with waves present in the electromagnetic spectrum, it has an associated equation that characterises the power received by the laser after a reflection of the light rays incident on the surface of the desired target. This equation takes into account the losses that occur during the round trip and the effect of the photodetectors that receive the light.

$$P_r = E_p \cdot \frac{c\mu A_r}{2r^2} \cdot \beta \cdot T_r \tag{4.1}$$

The equation models the amount of power received P_r by the sensor optics at a distance r from an object. In addition, the equation contains:

- E_p: total energy of a transmitte pulse laser.
- c: speed of light.
- μ: overall system efficiency.
- A_r: area of receive aperture at range.
- β: reflection of the target's surface. It depends on the surface properties and the incident angle of the light beam.
- T_r: transmission losses due to the transmission medium.

As can be seen, the power received from the measurements decrease quadratically with respect to the distance r, because objects hundreds of metres away will have weaker and more unreliable detections for the system than detections at close range. In sector jargon, objects at greater distances are said to be darker. In the parameter T_r the meteorological effects in the surroundings of the vehicle can be reflected. In case of rain or heavy fog, water in suspension in the environment scatters the light photons and interferes with their propagation. In this case, this parameter would decrease.

Types of LiDAR

Within LiDARs that base their operation on the Time of Flight (TOF) concept, we can find a wide range of categories depending on their construction or operation principles. A summary of the following categorisation can be seen in Figure 4.3 with an example of a commercial LiDAR sensor reference on the market for each of the types.

1. Scanning LiDARs

Scanning systems are designed to transmit beams of light capable of scanning a vast area rapidly. Scanning systems can be divided into two subcategories according to the nature of

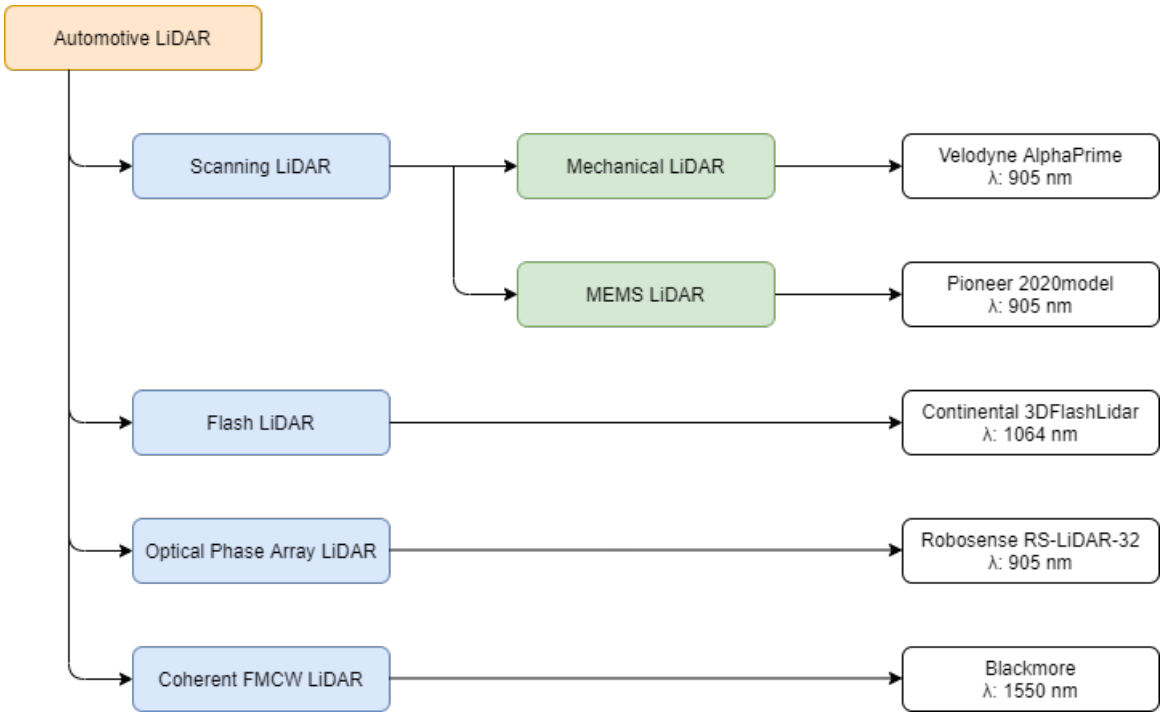


Figure 4.3: Types of LiDAR diagram
Source. Own elaboration

their construction principle, as they can be mechanically rotating or by means of mirrors and solid-state circuits. Solid-state systems refer to scanning systems that have no moving parts and are preferred in the autonomous vehicle industry because they are expected to perform better than mechanical ones and to lower the cost of such a sensor, one of the major disadvantages for their commercialisation.

- **Mechanical**: Rotation or mechanical gyro-based solutions are currently arguably the most popular LiDAR scanning systems in the automotive industry. They are based on a motor that rotates a mirror prism that generates light beams in all directions to achieve a near 360° horizontal field of view for a complete scan of the scene. In addition, it is possible to obtain a full 360° view if the base of this sensor also rotates.

  However, state-of-the-art sensors try to reduce the number of moving parts of the sensor as much as possible and achieve the 360° view by adding more triggers. The Velodyne VLP sensor series (which is the most celebrated solution on the market, with its HDL64 and VLS128 models) uses an array of laser diodes and photo-diodes to increase the density of its point clouds.

  A mechanical rotation system offers a high SNR (signal-to-noise ratio) and a large field of view, but at the price of being fragile for the high vibration conditions that an autonomous vehicle is used to.

- **Solid-state**: Scanning systems based on solid-state circuits are often referred to in the industry as Microelectromechanical systems microscanning or MEMS. They are systems that are composed of mechanical and electromechanical parts embedded in silicon chips.

MEMS mirrors are mirrors embedded in a chip and are rotated by the balance of forces: an electromagnetic (or Lorentz) force produced by a coil around the mirror and an elastic force produced by the torsion of the mechanical parts about the axis of rotation, with both forces opposing each other. As the parts are embedded in a chip, these rotations are microscopic and can work with modes of operation that offer large deflection angles and work at high frequencies.

Large mechanical parts are therefore being replaced by electromechanical equivalents that are controlled by voltages. Like its mechanical counterpart, this sensor is also fragile to vibration and does not work well in extreme temperature conditions (temperature requirements for vehicle applications demand operability at -40°C).

2. Flash LiDARs

A flash lidar is a solid-state lidar that completely eliminate the mechanical and moving parts found in scanning lidar and operates like a flash camera. The concept was originally born for space applications and was used for landing spacecraft and satellites but the industry is also looking at this type of sensor as a viable possibility for Autonomous Driving.

It is said to operate like a camera because they are sensors that in a single shot are able to diffuse a large number of light beams through the medium, illuminating the entire available field of view of the scene at once. Then, a 2D array of photo-diodes is used to capture all the reflections of the lasers and after that the processing is done to generate the 3D point cloud. It is the same principle that uses an image to recreate a 2D image, using even similar CMOS technology devices.

This scanning approach eliminates the vibration problem of its predecessors. Moreover, as the first lidar based on true solid-state circuitry, it is easier to mass-produce and costs less. Its major disadvantage is the limitation in field of view and range. The 2D photodiode array for receiving the reflections usually limits the field of view to only 90°, while its counterparts have a 360° view. In terms of range, receiving a single capture limits the range to about 100 metres, as the intensity of the reflections is lower.

3. Optical Phase Array (OPA) LiDARs

It is another lidar based solely on solid-state circuitry that does not completely eliminate all moving parts. It has an operating principle similar to that of phased-array radar, whereby an OPA lidar is capable of triggering light beams by means of different types of phase modulators. This allows the speed of light to be changed by passing the beams through different lenses that change the laser trajectory and results in control of the optical wave-front shape and of the steering angles. Although it is seen as a promising product, there is no commercial solution on the market based on this technology at present.

4. Coherent Frequency Modulated Continous Wave (FMCW) LiDARs

FMCW-type lidar does not use a modulation based on the time-of-flight principle but chooses to use a coherent method. This consists of the use of chirps modulated with a variable frequency. By measuring both the phase and frequency of the chirp on return, both the distance to the target and the velocity can be obtained by applying the Doppler effect. More will be said about these concepts as this is a very common feature of the automotive radar currently in use, while in lidar, no product based on this technology has managed to establish itself as a reference in the market. As main advantages, this method reduces the computational burden of processing and generating the 3D point cloud, but the hardware becomes more complicated as digital signal processing circuits must be added to generate the chirps.

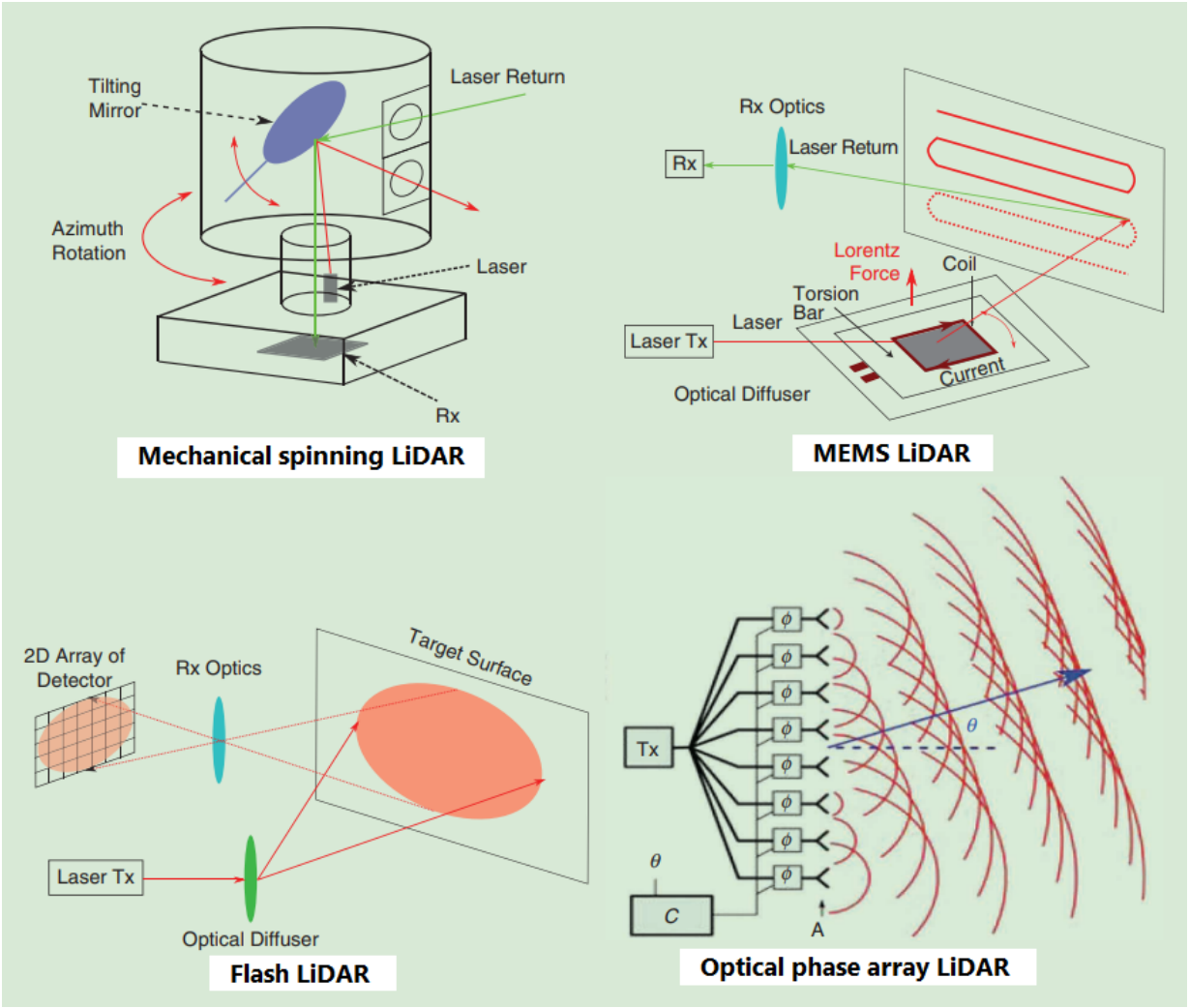


Figure 4.4: Lidar systems categorized by scanning approaches
Source. Lidar for Autonomous Driving. The principles, challenges, and trends for automotive lidar and perception systems [61]

Figure 4.4 shows some of the above operating principles, such as the mechanical rotation principle for spinning mechanic LiDAR, or the opposing forces for rotation on MEMS LiDAR, as well as the single-shot principle of LiDAR flash and phase reflection principle at OPA LiDARs.

Point Clouds

The data from the LiDAR sensor are received in multi-dimensional packets, where each element represents a point in space. These packages are called point clouds and a example of these can be seen in Figure 4.2. Each of the vectors has 3 or 4 dimensions depending on which LiDAR is used. In the first three dimensions the spatial location of that point is encoded, with each axis representing the X, Y, Z dimensions of space. In the fourth dimension the intensity of the measurement, denoted as I, is encoded. The intensity gives a normalised magnitude between 0 and 1 and quantifies the amount of energy that returns to the receiver after bouncing off the target object, as a ratio of the energy received to the energy emitted.

4.2.2 Fundamentals of RADAR

RADAR is an acronym for *Radio Detection and Ranging*. It designates a family of electromagnetic system whose primary purpose is the detection and location of objects in space. In addition to distances, it is also possible to obtain the altitude, direction and speed of possible targets. Radar measurements are possible thanks to the properties of electromagnetic waves that were introduced by physicists such as Maxwell, Hertz or Tesla in the late 19th and early 20th century. An electrical circuit generates electromagnetic waves that have the property of reflecting or bouncing off electrically charged surfaces. If this reflection returns to the emitter, it means that the surface it has bounced off is in the field of view of the sensor. Therefore, if electromagnetic energy is considered to travel at a constant speed equal to the speed of light, it is possible to determine the distance to the target by measuring the time it takes for the wave to be emitted and received again. As can be seen, the basic operating principle is the same as that of the LiDAR sensor, but in this case we have changed the length of the waves with which we work. Light waves are exchanged for radio waves, outside the visible spectrum.

The main strengths of radar-type sensors make them a very viable alternative or complement to visual observation sensors such as cameras. Radar is capable of operating day or night and its performance or accuracy does not depend in any way on the lighting conditions of the scene. Furthermore, it is also not dependent on weather conditions and is capable of operating in adverse weather conditions such as fog or rain. It is even able to penetrate some walls or a certain amount of snow cover. Thanks to this utility, a type of radar, the Ground Penetrating Radar, appears, which serves as an ADAS system for autonomous driving assistance in climatic environments where snow is prevalent. Nowadays, in addition to GPRs, the most commonly used sensor types in the Autonomous Driving sector are FMCW radar and 4D Imaging Radar, which offer additional resolution and are the most innovative.

Historically, Nikola Tesla was the first to propose the use of electromagnetic waves for the detection of moving objects in 1900. However, it was not until 1930 that British industry began to make serious proposals for radar systems. Early proposals included radar-equipped ships and aircraft to detect other items in their vicinity and improve navigation of both, one by sea and the other by air, respectively. However, it was not until the outbreak of World War II that the British decided to surround the UK coastline with radar sensors (Figure 4.5) to detect a pos-

sible German air invasion. From this point on, all the powers involved in the war, both Allied and Axis, began to invest in and develop radar equipment. Over the following decades, navigation and traffic control in the maritime and air domains continued to drive the development of radar systems, with radar being one of its main areas of application. It has also been used for space applications, supporting navigation. Outside of driving and navigation, the meteorological sector also makes intensive use of radar systems for measurements enabling weather forecasts to be made. From 1990 onwards, it began to be equipped in the field of ground-based driving, and today it is part of a large number of ADAS.

During this section, the principles of operation and the main elements of a radar will be reviewed. Basic concepts such as the Radar Equation or the Radar Cross Section will be covered and finally, the three types of radar mentioned in this introduction will be reviewed.

Figure 4.5: Radar Station on the East Coast of Britain (1946)
Source. iwm.org.uk. How Radar Gave Britain the Edge in the Battle of Britain

Radar Block Diagram and Operation Principles

As radar systems have been developed for more than 100 years for a wide variety of applications, the diversity of designs is very rich. However, it is possible to present a number of elements that must be present to achieve the most basic mode of operation of a radar.

- **Wave Generator**: modern radars operate with complex signal waves. In order to generate these signals, such as sinusoid signals with variable frequency, these circuits are placed into the system.
- **Transmitter**: an electronic circuit that produces a short pulse or a continuous high power radio frequency signal that is propagated through space by the use of transmitting antennas. Its behaviour is governed by the synthesizers or modulators that are in charge of generating the waves.
- **Receiver**: are electrical or electronic circuits capable of amplifying and measuring the very weak signals arriving at the device after bouncing off a surface in the radar's field of view. They must perform this task at very high frequencies and are supported by digital analogue signal converters.
- **Duplexer**: an electrical circuit whose purpose is to act as a switch between the different transmitting and receiving circuits so that they all operate together. In addition, it is necessary for the receiving antennas to be protected at the time of transmission since a reception of a newly transmitted signal could result in the reception of a very high magnitude of energy which would result in the destruction of the receiving device.
- **Antennas**: are the devices responsible for the emission and reception of electromagnetic waves from or required for the above-mentioned components. It therefore performs a conversion between electrical energy and electromagnetic waves in a bidirectional manner. They are normally metallic conductors.
- **Mixer**: specific components of continuous wave radars. It is an electrical circuit responsible for making electrical comparisons between the transmitted signal and the received signals to obtain additional information. Later on, it will be explained how this component allows to make use of the Doppler effect and to infer the speed of the detected object.

The combination of all these components in an electrical or electronic circuit enables the reception and emission of electromagnetic radar waves. These waves are then processed in Digital Signal Processors (DSP) to obtain information about the environment.

For the determination of the range or distance to a given object the most basic principle of operation is used: Time of Flight (TOF). This concept is now applied to the electromagnetic wave domain, as was done with LiDAR. Firstly, an impulse or a continue wave is sent from the emitter. Then, the energy that returns to the radar receiver, which is known as an echo, is used to compute the distance to the target as:

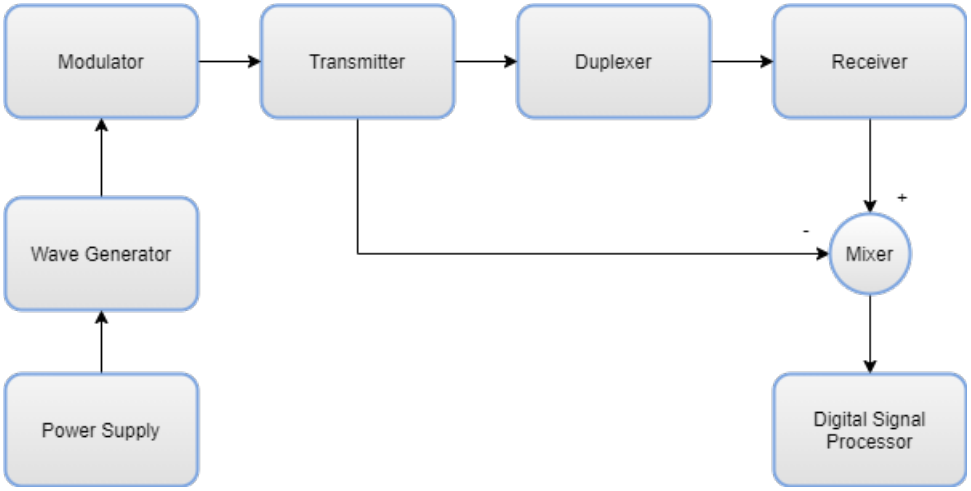


Figure 4.6: Universal Block Diagram of a FMCW radar
Source. Own elaboration

$$d = \frac{c_0 \cdot t}{2} \tag{4.2}$$

In this equation, already seen in Section 4.2.2, it is noted that the distance to the target (d) can be obtained as the product of the speed of light (c_0) times the wave travel time (t) divided by 2.

To determine the direction or angle from which the energy has been received from the object, it is necessary to know the directivity of the antenna. This directivity is the ability to concentrate a particular amount of energy coming from one direction. By measuring the direction in which the antenna points when the echo is received, the azimuth and elevation angles of the target can be determined.

Another very important concept when talking about radar is range resolution. It is defined as the ability to distinguish targets that are located very close in space or direction. The range resolution depends on the width of the transmitted pulse and the bandwidth of the radar device.

Radar Equation

The Radar equation is a well-known expression representing the physical dependencies of radar transmit power. It relates the maximum range to the characteristics of the transmitter, receiver, antenna, object and environment. By bringing together so many concepts, it becomes the typical way to express sensor performance and, in the initial stages, to understand the principles of operation mentioned in the previous Section.

To start with this development, ideal conditions for the propagation of electromagnetic waves are assumed, i.e. no scattering. If the emitter is an isotropic radiator, the energy will propagate in all directions, forming a sphere around it. As the distance increases, the energy loses power, so the two quantities are inversely proportional. With this, the radiated power density in all directions can be obtained.

$$S_u = \frac{P_t}{4\pi R^2} \tag{4.3}$$

In the equation, S_u is defined as the Omnidirectional Power Density, P_t is the transmitted power and R is the distance from the transmitting antenna to the evaluated object or point.

To focus the emissions of the isotropic radiator in the desired direction, the antennas are used. As these are directive, they will act by applying an antenna gain (G) that will focus the energy in the desired direction.

$$S_g = S_u \cdot G = \frac{P_t \cdot G}{4\pi R^2} \tag{4.4}$$

Where S_g is the Directional Power Density and G is the antenna gain.

When the emitted power intercepts with the target, it will re-radiate the power in multiple directions. These directions will depend mainly on the geometry of the target. To define this phenomenon, the concept of Radar Cross Section (σ) appears, which will be explained later and has units of area. Therefore, the reflected power P_r will depend on the directional power density multiplied by the RCS of the target.

$$P_r = \frac{P_t \cdot G}{4\pi R^2} \cdot \sigma \tag{4.5}$$

Analogous to what was seen for the emitter, the Energy Density at the Receiver S_e is defined as follows:

$$S_e = \frac{P_r}{4\pi R^2} \tag{4.6}$$

Since the Energy Densities have been defined at the sites of interest, reception and emission, it is time for the power to go back to the radar receiving antenna which will capture a part of the Energy Density at the Receiver S_e depending on the effective area of the antenna (A_e). This magnitude is known as Received Power (P_e).

$$P_e = S_e \cdot A_e = \frac{P_t G \sigma A_e}{4\pi R^2} \tag{4.7}$$

The last step is to combine the expressions for the reflected power (P_r) and the received power (P_e) by means of a system of equations to consider the outgoing, from the antenna to the target, and the return, from the target to the antenna. The result of solving the system of equations is:

$$P_e = \frac{P_t \cdot G \cdot \sigma \cdot A_e}{(4\pi)^2 \cdot R^4} \tag{4.8}$$

This equation is also used to determine the maximum range or distance at which an object can be detected. For this, it is assumed that this occurs when the received power is equal to the minimum detectable power (P_{min}) and the equation for the range is cleared.

$$R_{max} = \sqrt[4]{\frac{P_t \cdot G \cdot \sigma \cdot A_e}{(4\pi)^2 \cdot P_{min}}} \tag{4.9}$$

This would have provided the fundamental form of the Radar Equation, which can be complemented with phenomena that represent the losses as a function of the environment, which is outside the scope of this analysis. It should be noted that two parameters of vital importance for the correct choice of antennas, the antenna gain (G) and the effective area (A_e), have been introduced.

Doppler Effect

Doppler effect or Doppler shift is a physical phenomenon by which an electromagnetic wave suffers a frequency variation during its journey through the air between the transmitter, receiver and return. A very common example of this effect can be seen in ambulance sirens. During the approach, the frequency is higher and the sound increases while it decreases as a function of distance and direction as the object moves away.

This phenomenon is one of the strengths of radar and explains why it has always been so well suited to navigation applications, whether by air, sea or land. Thanks to this effect, it is possible to obtain the speed of objects in the sensor's field of view. More scene information with a minimum source of error, as each detection has a speed associated with it. This is why radar point clouds are also 4D, the 3 dimensions of spatial location (normally expressed in angular coordinates) are maintained, although unlike LiDAR, the fourth dimension corresponds to the speed.

The fundamental expression describing this effect comes from Classical Physics and is:

$$f = \frac{v + v_r}{v + v_s} \cdot f_0 \tag{4.10}$$

Where f is the observed frequency, v is the speed of the waves in that environment (normally is the speed of light c), v_r is the receptor speed, v_s is the source speed and f_0 is the emitted frequency.

In the field of radars that include Doppler effect, it is interesting to express this effect as a function of frequency variation, since it is known as bandwidth in these devices and is one of the parameters that put constraints on the design of radar systems.

$$\Delta f = \frac{2\Delta v}{c} \cdot f_0 \tag{4.11}$$

This allows the speed to be obtained natively in the processing of the digital signal received at the receiving antenna, a fact that supports the minimum source of error. If this calculation were to be made using the Time of Flight (TOF) concept, it would depend on the time derivative of its expression, a mathematical operation that would have to be implemented in the on-board

computer and would have its restrictions in terms of computation time and accuracy of the calculation.

Radar Cross Section

Radar Cross Section (RCS or σ) is a parameter specific to objects that have the ability to reflect electromagnetic waves. It measures how detectable such an object is by radar or, in other words, the electromagnetic signature of an object. As mentioned above, it has units of area.

The main factors influencing the RCS magnitude dependence are: the material of which the target is made, the relative size of the object with respect to the radar wavelength, the absolute size of the target, the angle of incidence of the wave, the angle of reflection and the polarisation of the waves. Therefore, it can be summarised that it depends mainly on geometrical factors at the time of measurement and on the size and materials of the radar and the target.

Depending on this parameter, more or less reflected energy is needed for an object to be detected. In order to test systems or to make an object more detectable, so-called corner reflectors appear. These are retroreflectors consisting of three perpendicular surfaces with intersecting flat surfaces. Their geometry causes the incident rays to bounce inside them three times, once per flat surface, resulting in a reversal of direction. This allows the amount of energy reflected and received at the receivers to be much greater than it could be from any other random geometry in the environment. In the field of aerial or maritime navigation, it is very typical for buoys or reflectors to have corner reflectors incorporated to facilitate their detection. A sample of the RCS of several typical targets can be seen in Table 4.1.

Targets	**RCS** [m^2]	**RCS** [db]
bird	0.01	-20
man	1	0
cabin cruiser	10	10
automobile	100	20
truck	200	23
corner reflector	20379	43.1

Table 4.1: RCS for point-like targets
Source. radartutorial.eu. Radar Cross Section

Automotive Radars

The radio spectrum is a part of the electromagnetic spectrum that contains the frequency range from 30 Hz to 300 GHz. All electromagnetic waves emitted in this range are known as radio waves. Given the advantages and strengths offered by this technology and in order to prevent interference between different users, a hierarchical division of the radio spectrum is necessary in which some bands are restricted for some applications or others.

However, the number of technologies using radar is very wide and as far as this work is concerned, we are only interested in the bands where automotive radar is used. Currently,

two types of automotive radars for ADAS can be distinguished: short-range and long-range. Short-range radars typically operate at frequencies around 24 GHz, with a wavelength of a few centimetres. As can be seen in Figure 4.7, these radars are in the K frequency band. In the case of long-range radars, these usually operate with frequencies between 76 GHz and 81 GHz, this bandwidth being one of the key parameters to make a system capable of adapting to different configurations. Also in Figure 4.7, it can be seen that these radars operate in the W frequency band.

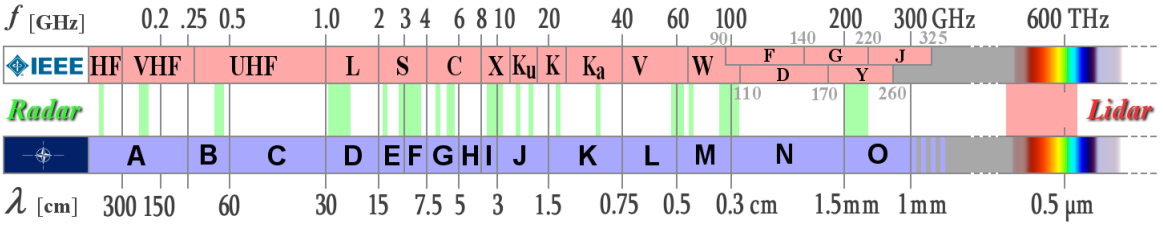


Figure 4.7: Waves and frequency ranges used by radar.
Source. radartutorial.eu. Waves and Frequency Ranges

To analyse the context of an autonomous driving scene, the concept of sensing must be much more precise than in the classic concept of military ground or air navigation. In this case, every single agent appearing in the scene can pose a danger to the driving. This is why the study on the improvement of radar resolution, both in the horizontal and vertical dimension, is encouraged. It can be said that the trend in research and industry on radar enhancements is focused on improving sensor resolution. This would make it possible to better discern objects at very short distances and not be classified as a single object. Being able to differentiate between objects scattered across kilometres is a totally different scenario than the one proposed by autonomous driving. Pedestrians within a few centimetres of each other or vehicles approaching at high speed and at close range are part of a much more complex scenario.

With these considerations in the paradigm shift from the traditional scope, the following challenges for automotive radar in the coming years can be distinguished [62]:

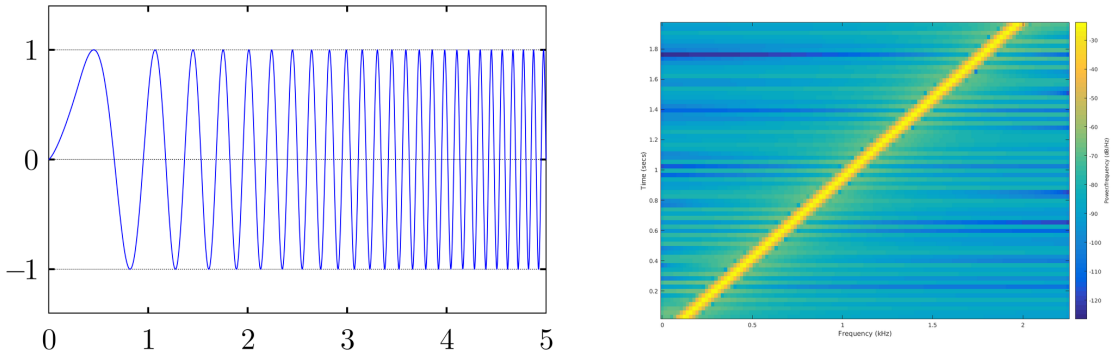
- **Scene variety**: as already introduced, the urban, rural or motorway scene in which automotive radar can operate is very diverse and can even be full of non-RCS-friendly objects. This is why radars are needed that can detect, locate, track and classify objects ranging from slow-moving children to animals in car parks to speeding vehicles. This requires optimal optimisation of parameters such as waveform, chirp duration, duty cycle or frame size and others.

- **High resolution**: a detailed scan of the environment is necessary to provide information on the shape and context of the detected objects. This is achieved with resolution. High resolution is needed for parameters such as range, Doppler, azimuth and elevation to achieve LiDAR-like performance. Obviously, the increase in resolution implies an increase in the size of the data packets sent by the sensor, which results in an increase in computational time and requires new, more efficient algorithms with high-dimensional data.

- **Clutter**: is the name given to unwanted echoes that are received by the sensor. At short operating ranges, it is very typical to receive clutters from the radar antennas themselves or from elements forming the track, such as the ground or walls. Any geometrically shaped element with perpendicular surfaces could give a false positive and this is why future radar challenges must be studied to resolve this.
- **Interference**: in a world full of autonomous vehicles, all equipped with radar sensors operating at the same frequencies, interference is a source of error that can lead to erratic sensor behaviour. There are three main sources of interference in this case: the sensor itself, other radar sensors installed in the vehicle, and radar sensors installed in nearby vehicles. The first ones mainly affect objects at very close range. This leads to an increase in computational costs since the processing algorithms must implement elements to mitigate these unwanted behaviours.

Frequency-Modulated Continous Wave Radar

FMCW radars are a special type of sensors that radiate power continuously and base their range measurements on the frequency variation that occurs in the wave from the time it is emitted until it returns to the receiver. Therefore, the main element of these radars is a type of signal called chirp. A chirp is a signal that follows a sinusoidal shape whose frequency varies linearly with respect to time.



(a) Chirp in time domain (b) Chirp in frequency domain

Figure 4.8: Linear chirp representation in time and frequency domain
Source. en.wikipedia.org. Chirp

A chirp is characterised by a start frequency (f_c), a bandwidth (B) and a time duration (t_c). Another parameter is the slope (S) (visible in Figure 4.8b) which defines the growth of the frequency with respect to time. As this increase is linear, the target parameters (range, Doppler...) are obtained by a signal comparison performed in a mixer. The result of the comparison of two sine waves is another sine wave with a frequency and phase equal to the differences of the two signals.

As the key parameter for range and Doppler determination is frequency, these signals are usually worked with in the frequency domain. For this purpose, intensive use is made of the

Fourier transform. An ideal sinusoid in the time domain produces an impulsive signal in the frequency domain. Two objects at the same distance will produce a frequency impulse for the same magnitude and this is where the Range Resolution comes into play. To solve this problem, the radar bandwidth must be increased, as defined by the following expression.

$$d_{res} = \frac{c}{2B} \tag{4.12}$$

Signals in the frequency domain are usually complex numbers, because they are composed of amplitude and phase. This is used to estimate the velocity of an object by using 2 chirps. If two consecutive chirps are transmitted divided by a time T_c, the range-FFT plot will have both peaks at the same location but with different phases. The phase difference (ω) corresponds to the motion of the object.

$$\omega = \frac{4\pi v T_c}{\lambda} \tag{4.13}$$

Following this principle, there will be no ambiguity for pairs of chirps where $\omega < \pi$. Therefore, the maximum relative speed that can be measured by 2 chirps separated by a time T_c is:

$$v_{max} = \frac{\lambda}{4T_c} \tag{4.14}$$

But if you want to measure the distance of two equidistant objects approaching the radar at different speeds, you will not be able to tell which of the phases belongs to which object. You will need to transmit an equi-spaced sequence of N chirps, which is called a frame. An FFT for a sequence of phasors corresponding to the range-FFT peaks solves the problem for two objects. This new representation is called the Doppler-FFT.

Thanks to this new Doppler-FFT, a measure of its capability can be obtained, which will correspond to the Velocity Resolution. This will be inversely proportional to the time of the frame (T_f) and is expressed as:

$$v_{res} = \frac{\lambda}{2T_f} \tag{4.15}$$

The last measurement that remains to be deduced is the angle at which the objects meet. To determine the angle of the object it will be necessary to make use of at least 2 RX antennas. This is because the differential distance of the object to each of the antennas translates to a phase shift in the Doppler-FFT plot, also known as 2D-FFT. Therefore, this difference between the patterns from each of the RX antennas is exploited. The measured phase difference can be used to estimate the angle of arrival of the object.

$$\theta = \arcsin \frac{\lambda\omega}{2\pi d} \tag{4.16}$$

Angle estimation is most accurate at θ close to zero and estimation accuracy degrades as θ approaches to 90 deg. At the same time, another concept is defined: Angular Field of View. It gives a measure of the maximum field of view that can be covered by two antennas which are separated by a distance d. As can be seen in the following expression, a spacing of $d = \lambda/2$ gives the largest AFoV.

$$\theta_{max} = \arcsin \frac{\lambda}{2d} \tag{4.17}$$

Analogous to velocity, if you want to obtain the angle of arrival of two equidistant targets approaching the radar at the same relative (Doppler) velocity, you will need to use an array of N antennas. An FFT on a sequence of phasors corresponding to the 2D-FFT peaks would solve this problem. This new plot is called Angle-FFT. In addition, the concept of Angle Resolution appears, which is the minimum separation in terms of angle for two detections to appear as separate peaks in the Angle-FFT plot.

$$\theta_{res} = \frac{2}{N} \tag{4.18}$$

Thus, while velocity estimation takes advantage of the phase shift between chirps separated in time, angle estimation takes advantage of the phase shift between chirps taken at the same time interval but separated in space, i.e. from different RX antennas.

As conclusions:

- **Range resolution**: is directly proportional to the Bandwidth (B) of the chirp and automotive radar can span a large bandwidth (4 GHz).
- **Velocity resolution**: is inversely proportional to the frame time (T_f) so it can be improved with the correct hardware to minimums.
- **Angle resolution**: is directly proportional to the number of RX antennas. This causes circuits to increase in complexity, cost and silicon area required, making it the most difficult of the three to improve.

This study leads us to conclude that Range and Velocity resolution are native strengths of the radar sensor.

4.2.3 Exploring Sensor Fusion of LiDAR and RADAR

One of the main objectives of this work is to explore and exploit the benefits and disadvantages of using a perception system based on sensory fusion of LiDAR and RADAR. Therefore, before obtaining conclusions (7) of the experiments carried out during the work (6), it is of great interest to carry out a previous theoretical analysis of this tandem of sensors in order to be able to make a further comparison and analysis with better criteria and rigour.

At a glance, it can be seen in Figure 4.7 that both sensors operate on the electromagnetic spectrum but at different frequencies. While LiDAR works in the visible light spectrum, RADAR works with a shorter wavelength. This means that many of its operating principles are very similar and that there are several types of LiDAR and RADAR that meet the same concepts, but on the contrary, it also makes one sensor perform better in some tasks than the other. This gives both sensors a different role in forming a tandem in sensory fusion tasks.

Enumerating and comparing both sensors in the following characteristics:

- **Range**: Both sensors stand out because they are able to detect objects at long distances. However, comparing the two, it should be noted that RADAR obtains higher resolutions (150-250 m) than LiDAR (100-200 m) at the price of lower performance at short distances.
- **Resolution**: One of the biggest disadvantages of RADAR is its lack of resolution, both spatial and angular, and this is perhaps the biggest problem it needs to address if it is to become an essential sensor for a perception suite in autonomous driving. This problem has already been solved for LiDAR, which has an excellent ability to discern closely spaced objects. The higher resolution of LiDAR means that the point cloud obtained is richer in number of detections than that of RADAR.
- **Works in dark or bright**: As both are active sensors (emitting energy in the data capture process), they do not suffer performance drops in low-light conditions, so they can perform at their best in such adverse scenarios.
- **Works in snow, fog or rain**: In this case, weather conditions that include the suspension of water in the air, either in liquid or gaseous form, make LiDAR perform worse than under normal conditions. As seen in this chapter, light rays are diffused and their direction changes as they pass through water molecules. This is something that does not happen in RADAR, which works at full capacity in the presence of water in the environment.
- **Provides color and contrast**: None of both sensors is able to encode information about the colour of the scene, as the camera does, since electromagnetic wave-based detections encode other types of information. In order to achieve a perception system that senses the colour of the scene, the inclusion of the camera in the sensor fusion would be necessary.
- **Detects speed**: It is possible to detect the speed of detected objects with both sensors. However, while LiDAR requires processing involving inference based on several time-spaced and tracked detections, RADAR is able to provide a native measure of the relative velocity of the object based on the Doppler effect. This eliminates errors and allows for a more accurate detection.

Therefore, a perception system based on LiDAR and RADAR has as major advantages a large range, where the early RADAR detections serve for an intelligent processing of the LiDAR detections. In addition, it complements the scene information with a native RADAR speed that improves the tracking tasks of the system.

Chapter 5

Software Technologies and Frameworks

A dream doesn't come true by magic, it takes sweat,
determination and hard work.

Colin Powell

5.1 Introduction

The main objective of this Chapter is to study the software technologies and frameworks used during the development of the this work. This will continue the task of contextualising the reader in the field of research in autonomous driving systems, since all the technologies to be mentioned are in one way or another at the forefront of research in this sector and in Software Engineering.

During the Chapter, a Section will be given to each of the technologies involved in the work. However, it is possible to discern three categories to classify all the tools to be used. The first category consists of the ROS robotics software development framework and the CARLA autonomous driving simulator. The union of these two tools through the CARLA ROS Bridge package constitutes one of the most powerful set-ups for the development, training and validation of autonomous driving systems.

The second category is made up of three tools that enable the development of large-scale open source software projects. Ubuntu Linux, Git and Docker make up a trinity that facilitates the development of this type of applications. Ubuntu Linux is the Linux distro par excellence, while Git acts as version control software and Docker offers software containers to quickly deploy application releases.

The third and last category is composed of two of the most famous Python packages for the development of Machine Learning based applications will be presented: Scikit-Learn and Open3D, the latter being a very new tool.

5.2 Robot Operating System (ROS)

Robot Operating System or **ROS** [63] is a free and open source ecosystem or framework for the cooperative development of robots. Despite what its name indicates, it is not an operating system but a set of middleware tools that can act as a pseudo-operating system that provides developers with basic functionalities that allow them to abstract from the lower layers of development and build a complex architecture.

In other words, ROS is a framework for writing robot software. A collection of tools, libraries and conventions that aims to simplify the creation of complex and robust robot behaviours. ROS was created in the 2000s by Stanford University and Willow Garage in response to the need to establish a common set of guidelines when developing robots. This comes at a time when the terms robotics and artificial intelligence are beginning to resonate in engineering companies and universities around the world.

Three main elements can be distinguished in the ROS ecosystem:

- **Framework and tools**: this section contains all the libraries and dependencies that support the framework. In addition, ROS provides the developer with a set of tools that will facilitate the development of his product. This is done following the philosophy of the UNIX operating system, large and complex architectures can be created from a composition and parallel work of many small and simple programs. Among the tools we can find `catkin` or `colcon`, tools with which to compile ROS1 and ROS2 packages, respectively; `rviz`, a tool to visualise and draw system messages in the environment for visual debugging; `rqtgraph`, a visual for obtaining graph representations of the system; or a set of tools to launch nodes in a simple way, together or accessing directly through the terminal to the elements in which the communication takes place, among others.

- **Client libraries**: these libraries are implementations of the functionality provided by the operating system and are presented in a developer-friendly way so that the developer can make use of them in the form of packages. For the ROS1 version, there are official client libraries for `C++` and `Python`, `roscpp` and `rospy`, respectively. These two wrap the content of the framework independently, to take advantage of the features of each language. However, the community has grown considerably and there are other implementations for other languages that demand the framework, such as `roslisp` or `rosjava`. For the ROS2 version, it has been decided to create a common library, `rcl`, written in C language. This ensures that all implementations in different languages contain the same functionalities without language-specific restrictions, since the new client libraries are built from wrappers or bindings, just as `rclcpp` and `rclpy`, the `C++` and `Python` client libraries, are. This makes it easier for the community to create client libraries for other languages in a homogeneous way.

- **Packages and libraries**: packages are the entity in which ROS-based software projects are compiled. They can be for public or private use. However, thanks to the open source nature of the framework, once the two previous elements are available, it is possible to

develop packages and libraries that can be made available to the community to be reused in other projects. They can be maintained by programmers from the community or by large companies that offer everyone the possibility of incorporating their improvements or features in the development of their software robot projects. There are a large number of packages that streamline recurring tasks in robot development: `ros_control` for controller development, `ros_navigation` for navigation tasks, `MoveIt!` for robot arm movement, `ros_plan` for executing planning tasks and many others.

The programming paradigm and philosophy of the ROS framework is based on the peer to peer concept. ROS-based systems are composed of small programs called nodes that are interconnected with each other through communication mechanisms called messages (topics, actions or services) in a distributed manner. This results in the ability to create systems that are scalable and that respond satisfactorily to a large flow of information and communication between programs.

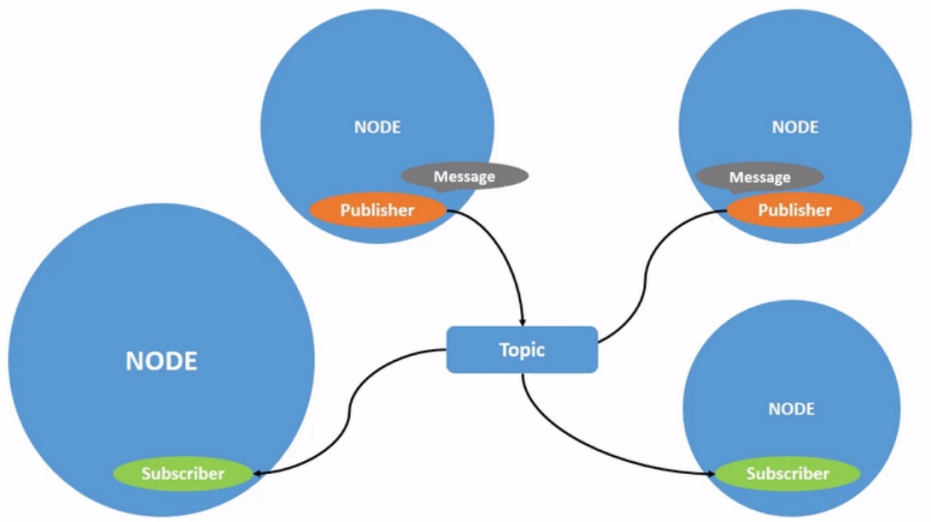
Type	Used for
Topic	One-way communication, especially if there might be multiples node listening. Suitable for streams of sensor data.
Service	Simple request/response interaction, such as asking a node's current state or processing an image once.
Action	Complex request/response interaction in which it is not instantaneous and feedback is needed.

Table 5.1: Elements of communication in ROS: topics, services and actions
Source: Programming Robots with ROS (O'Reilly)

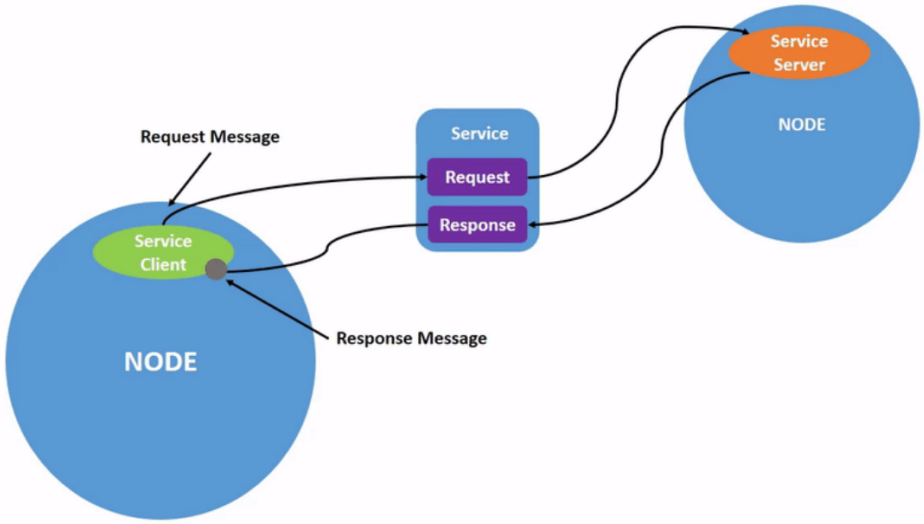
Therefore, nodes can take the role of publishers or subscribers if they are going to send messages via topic or they can request or receive messages if they opt for the action or service form of communication. In addition, in topics, more than one node can publish in it or can be subscribed to its information. Similarly, in actions and services, more than one node can take the role of requester or receiver. Figure 5.1 shows possible system configurations that operate around these concepts, and with various casuistries in terms of number of nodes and types of communication. In addition, Table 5.1 shows which of the above-mentioned message types is most suitable for each situation.

A supervisor role also exists and is carried out by the ROS Master node. This is the primary node and is responsible for setting up the system and weaving the network of nodes required by the application. The nodes would not be able to meet each other, exchange messages or invoke services or actions without the presence of the Master. It is a figure that must be directly invoked in ROS1, while in ROS2 it is invoked automatically and transparently to the user.

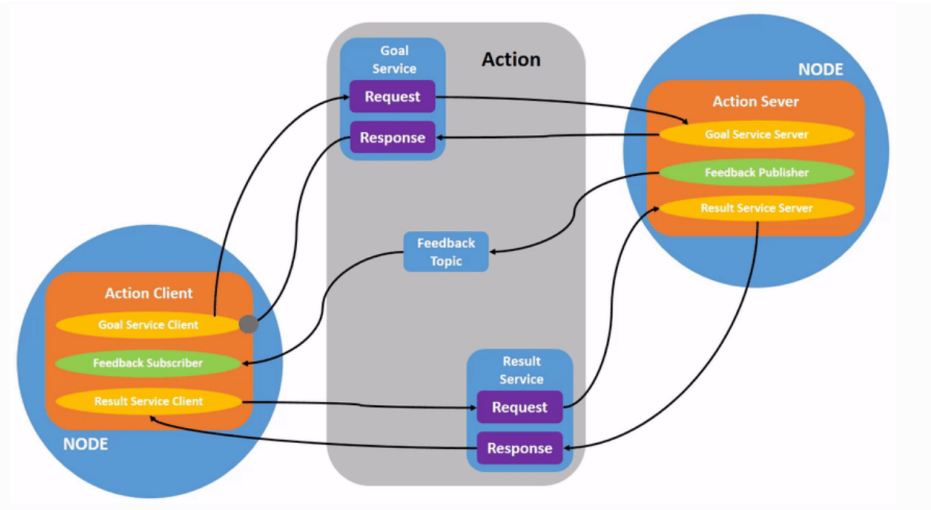
Another important figure in ROS are the parameters. As their name suggests, they are entities that allow certain characteristics of the programme to be varied before the moment of running. Rosbag is another widely used tool. It allows you to record the flow of data and information that is being transmitted by topics or calls to services so that it can be replayed offline.



(a) Topics



(b) Services



(c) Actions

Figure 5.1: Elements of communication in ROS: topics, services and actions
Source. docs.ros.org. ROS2 Documentation

RVIZ Simulator

ROS Visualization or **`RViz`** is a tool that allows 3D visualization of the data and the information flowing through ROS messages: topics, services and actions. In other words, is a 3D visualizer for displaying sensor data and state information from ROS. It is a widely used tool in the field of perception and localisation, both for the development of robots and autonomous cars. In combination with this tool, ROS offers a series of utilities and graphical markers that allow a visual inspection of the environment, since it must be taken into account that an early detection of erratic or undesired behaviour in the code through this visual inspection can be of great help and boost the development of the application.

CARLA-ROS Bridge

The concepts and philosophy that characterise ROS and allow them to generate distributed systems have made the framework not only suitable for the development of robots, but has positioned it as a solid option in the autonomous driving sector. This is why CARLA, one of the most powerful open-source simulators in the industry (explained in detail in the next section), has developed a series of packages that act as a driver or bridge between its architecture and the ROS architecture: **CARLA ROS Bridge**.

This package provides support to make the data flow between CARLA and ROS as efficient as possible. Therefore, the user is provided with a series of scripts that transform data from CARLA format to ROS format or vice versa, since the communication between both entities is bidirectional. The union of both technologies constitutes one of the leading software suites in the field of development and validation of autonomous driving algorithms. In addition, the use of the integrated ROS tools, such as RViz, offers a very friendly and useful graphical interface for visual debugging and validation of the algorithms, as can be seen in Figure 5.2.

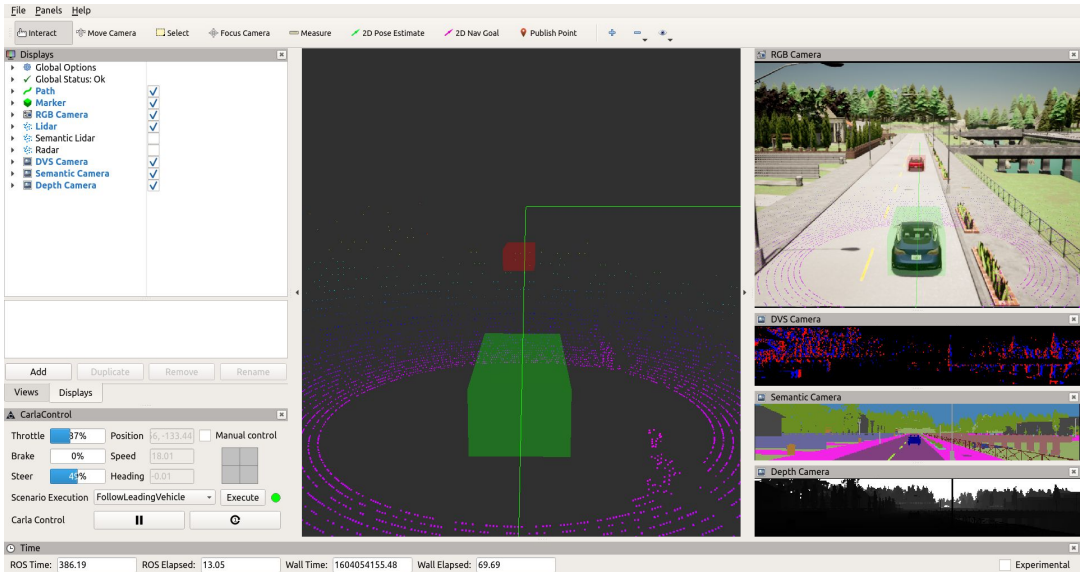


Figure 5.2: RViz visualization of a CARLA and ROS simulation
Source. `github.com`. ROS/ROS2 bridge for CARLA simulator

5.3 CARLA Simulator

CARLA [64] (Figure 5.3) is an open source simulator for research in the field of autonomous driving. Its name is an acronym for ***Car Learning to Act*** and it is currently listed as one of the solutions that make up the spearhead of research in this field. Developed through a collaboration between Toyota Research Institute, Intel Labs and Universidad Autónoma de Barcelona, it was conceived as a tool to support the development, training and validation of autonomous driving systems. All this following an open-source philosophy with a large amount of digital assets such as urban layouts, buildings or vehicles to be used freely for this purpose.

Figure 5.3: CARLA Simulator logo
Source. carla.org. CARLA Simulator

The simulator is designed to be able to offer flexible and scalable solutions. The platform supports different combinations of sensors, all types of environments, static and dynamic scene actors, map generation and a multi-client architecture via server so that multiple clients running on different nodes can control different actors.

CARLA is implemented on the video game engine Unreal Engine 4, owned by the company Epic Games. This graphic engine provides the simulator with realism in the rendering of the graphics and support for the inclusion of physics that increase the reliability of the simulation of the environment and the vehicle. Phenomena such as the friction of the wheels on the pavement or the inertia of the vehicle can be modelled in this simulator.

To interact with the CARLA world and modify it to suit the needs of a system, the developer is provided with an API written in Python on which many parameters of the simulation can be edited. These include traffic generation, the behaviour of pedestrians in the scene, the weather or illumination conditions of the environment and so on. One point of CARLA's great flexibility lies in the possibility of designing a completely customised sensor architecture. The user can choose between LiDAR, 3D with or without semantics, a multitude of cameras, depth, monocular or semantic segmentation cameras, radars and others for localisation such as GPS sensors. All this so that each project can be simulated in conditions as close as possible to the real-life set-up. In addition, each and every sensor has several parameters to edit, such as the field of view, the aperture angle of the cameras or the number of points collected per second in a point cloud.

On the other hand, CARLA's environment is composed of 3D models of objects such as buildings, traffic signs and structures, and dynamic objects such as pedestrians, vehicles or cyclists. All models are created while maintaining a trade-off between realism and simplicity in terms of textures and shapes to offer a simulation that is both light and detailed. To achieve this, use is made of the materials and lighting system of the graphics engine. After installation, CARLA provides a series of maps representing different urban, rural and motorway scenes on which the user can run their simulations. However, in one of its latest updates it received support for the generation of custom maps. It makes use of the RoadRunner tool and the OpenDrive standard with which multiple simulation tools generate scenarios for autonomous driving. Each scenario can be configured with a combination of lighting and weather conditions, generating a total of 18 possible weather and lighting pairs. This, together with the wealth of maps (as seen in Figure 5.4) and support for custom maps, makes for a wide range of possible scenarios to test.

In addition to the measurements from the sensors, CARLA provides the user with a series of metrics and data regarding the vehicle's status and its behaviour with respect to traffic rules. Parameters such as location and orientation in space with respect to the world, speed, acceleration and collision impact.

Figure 5.4: CARLA Town 11 Map
Source. carla.org. CARLA 0.9.12 Release

Another tool provided by CARLA is the ScenarioRunner. This tool allows users to define and execute different use cases that can occur during driving based on modular behaviours. All scenarios are defined through an interface written in Python and can make use of the OpenSCENARIO standard, which defines a set of guidelines to describe the dynamic content of the driving.

And it is not only possible to do simulations with sensors for the vehicle perception layer. Simulation plans for the planning and control layers are also built into the simulator. In this case, the rendering quality is lowered in order to provide a computationally efficient and as real-time as possible with a more detailed simulation in terms of traffic and road behaviours.

5.4 Ubuntu Linux

Ubuntu (Figure 5.5) is a Linux operating system distro that is based on Debian and is composed of free and open-source software components. It is one of the most popular Linux operating system distros on the market and this is reflected in the fact that it supports more than 55 languages.

Its software life cycle consists of releasing a new version every 6 months, while every 2 years, an LTS (Long-Term Support) version is released. The most recent LTS version is Ubuntu 20.04 LTS *Focal Fossa*, version that will be used in the development of this work, together with its LTS predecessor: Ubuntu 18.04 LTS *Bionic Beaver*. The company in charge of coordinating the project leading to the development of Ubuntu Linux is the British company Canonical.

Figure 5.5: Ubuntu Linux logo
Source. ubuntu.com. Ubuntu Linux Home Page

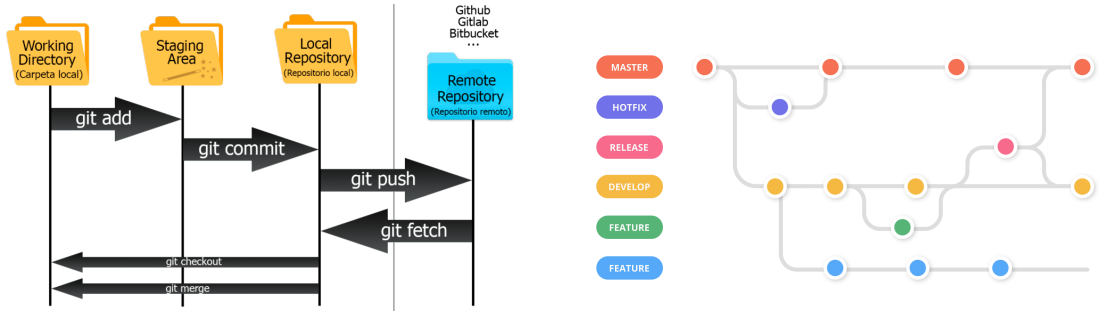
Ubuntu Linux is currently distributed in three official editions: *Desktop*, *Server* and *Core*. All three are well known in the research fields of robotics, autonomous driving and artificial intelligence. The first is intended to be run on personal computers, so it offers support for applications with graphical user interfaces, as well as a number of tools to enhance the user experience. Therefore, the *Desktop* version is the version used when developing or validating algorithms with the tools already presented in this Chapter. On the other hand, the *Server* version is a lighter version intended to serve systems composed of servers. The community saw an opportunity to implement this version on embedded systems by deploying lightweight and validated versions of the developed software. In response to this demand, Canonical responded to the needs of these sectors by releasing the *Core* version. It is a lighter version adapted for x86 and ARM architectures, typical in embedded systems. The purpose of this third version is to provide real-time processing on which safety-critical systems can be deployed.

Due to its open-source nature and high reliability, Ubuntu has become over the last few years the ecosystem in which the community has developed applications for the design and validation of software applications, and in the case in point, for the robotics and artificial intelligence sectors.

Companies involved in the development of autonomous driving systems or ADAS choose to use Ubuntu as the basis for the development of their applications. Some of these partners are: Toyota, BMW, Volkswagen or other brands such as Continental or Lyft. This trust has been placed because the operating system offers an ecosystem in which Artificial Intelligence and Machine Learning algorithms provide great performance using cloud computing. In addition, key applications for this development have been consolidated and have grown thanks to the system's facilities: ROS or Nvidia Isaac are examples of this.

5.5 Git

Git is a free, open-source version control tool designed to handle the smallest to the largest projects quickly and efficiently. One of its main advantages is that it is very lightweight and provides lightning performance. Therefore, it allows all developers involved in a project to coordinate to incorporate the required features by tracking the changes and progress of the files. It is a distributed version control system (DVCS), as it has a distributed architecture. Each programmer's working copy is a repository with a complete tracking system that holds the complete history of changes to each of the files. Git therefore facilitates code management and application development. A very common technique is to connect Git repositories with external repositories, the best known exponent of which is GitHub.



(a) Git Workflow diagram

(b) Git Branch structure

Figure 5.6: Key concepts of Git: repositories and branches
Source. atlassian.com. Become a Git Expert

Figure 5.6a shows a diagram of a developer's work with Git. The first element in the image is the Working Directory. It is nothing more than a local folder on the developer's computer where files are worked on and edited. After working on the project's text files, the developer can add them to the Staging Area using the `git add` command. This stage allows us to control the changes in the code (`git diff`) and add metadata about them. Then the Local Repository figure appears, which we move to using the `git commit` command. It is still a local file in which all the changes have been grouped and committed, having made the relevant revisions. The last figure is the Remote Repository. It performs the same functions as its local counterpart but stores all the files in the cloud. Local content is pushed into the Remote Repository using the `git push` command. This working methodology allows to have an exhaustive control over the changes produced in the file.

Figure 5.6b shows another of Git's main features, working through the use of branches. Branches are parallel lines of the project's code. They are used to make modifications to the project while making other versions of the fully functional code available. They are usually represented by such horizontal tree diagrams in which each of the points represents a `commit` (if working with local repositories) or a `push` (if the repository is remote). There is usually a main branch on which development or test branches are usually merged if development is successful.

5.6 Docker

Docker [65] is an open-source project that enables the development, shipping and running of applications. These tasks are automated through the concept of software containers, tools that provide an isolated environment in which the dependency that exists between the system and the architecture on which it has to be run is eliminated.

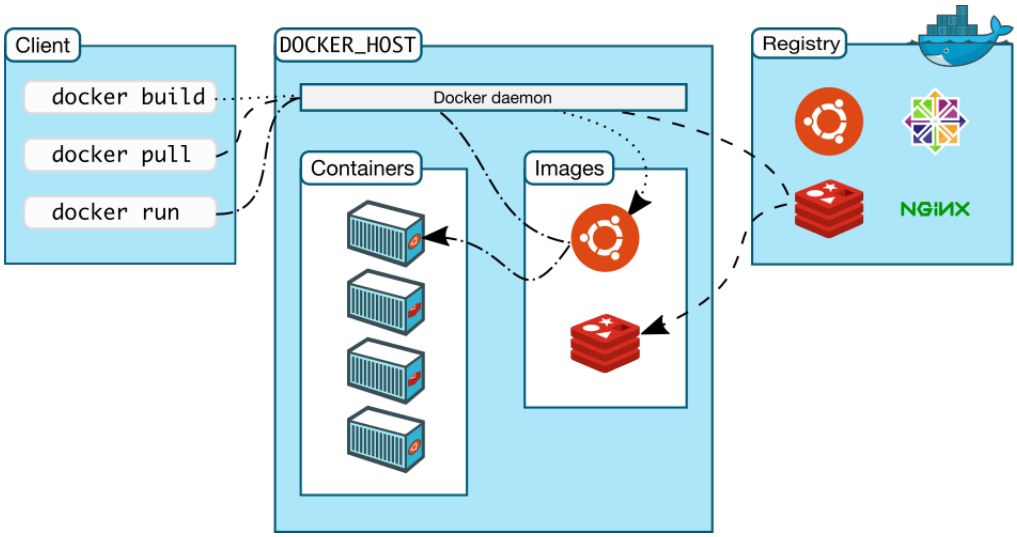


Figure 5.7: Docker architecture
Source. docker.com. Get Started with Docker

A container is defined as an executable instance of an image or software project. Therefore, the developer can insert all the required libraries and dependencies, source codes and frameworks needed inside the container and thus get a version that is able to port or migrate environment with ease. All container dependencies are defined in a file called dockerfile. This is very useful for projects where functionality needs to be tested in different environments, for example: an autonomous driving software project needs to have the versatility to be able to migrate from the simulation environment to the real set-up hardware suite.

Docker uses a client-server architecture, the diagram of which can be seen in Figure 5.7. The first element that can be seen is the Docker client, through which users interact with the architecture. All commands that are sent by the user are received and handled by the Docker daemon. The daemon is in charge of listening to the Docker API requirements and managing the rest of the Docker objects: containers, networks and volumes.

With all these features, Docker is a tool designed to improve the workflow of software engineers and developers. For example, they provide standardised environments for provisioning applications and services. They can be used for continuous integration and continuous delivery (CI/CD) methodologies. By separating development, test and production environments using containers, when a new feature passes all tests it can be put into production by updating the corresponding image. It is also useful for scaling and deploying applications, as they can be run on PCs, laptops and virtual machines. In this project, Docker is very useful to quickly migrate from the simulation environment to the real environment, carrying tools such as Nvidia CUDA, OpenCV, ROS and the dependencies they generate.

5.7 Scikit-Learn

Scikit-Learn is an open-source library designed to develop Machine Learning applications in Python. It provides developers with simple and efficient tools that allow them to create systems through predictive data analysis. In addition, it is built on top of some of the most famous libraries in the Python community: `numpy`, `scipy` and `matplotlib`. It is currently recognised as one of the leading libraries and frameworks for Machine Learning in Python, being a very common option for teaching and learning in the community. This means that there are a wide variety of courses and resources to learn how to use it and it is part of major projects in the industry.

It is structured in modules, the unit in which Python allows to separate the contents of a package. They are very useful to be able to separate some utilities from others. The main modules of Scikit-Learn are:

- **Classification**: allows you to identify to which category an object belongs. Among its main applications are the detection of spam in e-mail or image recognition. Representative algorithms in this category are Support Vector Machines (SVM) and Nearest Neighbors (NN).
- **Regression**: allows the prediction of the behaviour of an object against a set of attributes and their correlations. It is used to predict the responses of medicines and stocks through mechanisms such as Random Forest.
- **Clustering**: As we have already studied in Chapter 4, clustering is concerned with aggregating elements of related or similar characteristics into sets. It is used for the detection of objects in three-dimensional point clouds or for customer segmentation in marketing.
- **Dimensionality reduction**: this module addresses the task of reducing the number of random variables to be considered in a problem, improving visualisation and solving efficiency.
- **Model selection**: is a module responsible for comparing, validating and choosing the appropriate parameters and models for a given problem. It is used to increase accuracy through parameter tuning. Techniques such as grid search or cross validation and metrics are found in this module.
- **Preprocessing**: this module allows features to be extracted and normalised as a prelude to the application of algorithms. For example, it can be used to transform text in Natural Language Processing (NLP) applications.

As we have seen, Scikit-Learn provides programmers with a set of very interesting tools to solve problems using Machine Learning techniques. It also allows them to combine all of them to create sequential pipelines in which data is passed from one stage to another to achieve a final result. In this work, Scikit-Learn will mainly help in the understanding of clustering algorithms and their application to object detection in point clouds.

5.8 Open3D

Open3D [66] is a modern framework designed by Intel whose main purpose is 3D data processing. According to the authors' descriptions, the frontend offers developers a range of data structures and algorithms implementable in C++ and Python while the backend is carefully optimised for parallelisation. This combination enables the rapid deployment and implementation of complex algorithms that handle 3D data in two of the community's most popular languages.

Since the world can be modelled using 3D point clouds, they have been of great interest in areas such as robotics and geography. Therefore, Open3D arises from the need for a framework that can easily operate with this type of data, achieving a role analogous to that of the OpenCV framework in the field of artificial vision where 2D data is used, in which the developer has access to common and traditional processing algorithms.

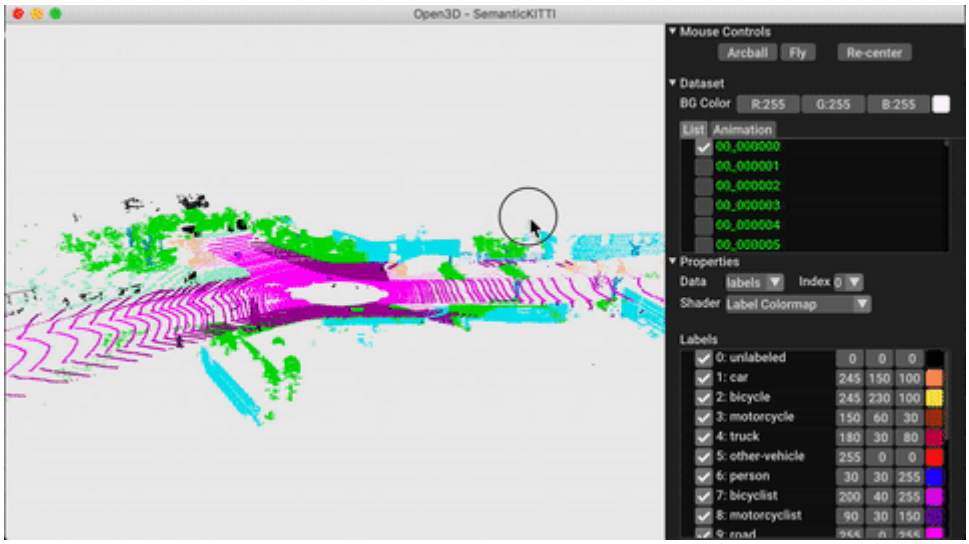


Figure 5.8: Open3D Visualizer for Autonomous Driving applications
Source. `github.com`. Open3D: A Modern Library for 3D Data Processing

Among the main features of Open3D it is possible to find: common data structures and processing algorithms for 3D data, scene reconstruction techniques, surface alignment techniques, 3D visualization tools such as the one seen in Figure 5.8, physically based rendering, 3D Machine Learning support with PyTorch and TensorFlow and GPU acceleration for 3D operations.

Chapter 6

Work Development

Intelligence consists not only of knowledge, but also of the ability to apply knowledge in practice.

Aristotle

6.1 Introduction

In this Chapter, the development of the main practical work of this project will be presented. To do so, it will make use of all the knowledge learned during the previous chapters. The practical part of this work is composed by a project of detection and tracking of multiple objects based on the LiDAR and RADAR sensors provided by the CARLA autonomous driving simulator will be presented in detail. It will be written in `Python` language and using popular libraries for working with point clouds and Machine Learning, `Open3D` and `Scikit-Learn`, respectively. The project will be integrated into the `ROS` robot software development framework and will be carried out in the context of the *Techs4AgeCar* research project.

As a note to the reader, it should be noted that most of the concepts mentioned in this chapter have already been explained in previous chapters, such as traditional techniques (4), state-of-the-art techniques (3) and software tools (5) used, so it is strongly recommended that you read them beforehand.

6.2 Multi-Object Tracking System based on LiDAR and RADAR

The main objective of this project is the development of an Object Detection and Tracking system for Autonomous Vehicle applications. It forms the core of the practical part of this work. This project will be written in `Python` programming language, the dominant language in the Machine Learning field. Therefore, use will also be made of two of the libraries and modules most widely used by the community for this purpose: `Open3D` and

`Scikit-Learn`. The repository for this project is publicly shared on `github.com`, with name: `santimontiel/lira-mot`.

6.2.1 Plane segmentation based on RANSAC

RANSAC [27] is an acronym for *Random Sample Consensus* and describes an iterative algorithm capable of estimating a mathematical model constituting a known geometric body, such as a plane or a sphere, in the presence of a noisy data set. It is therefore able to make an accurate and robust estimate of the parameters of a model, for example, the coefficients of a 2D line or a 3D plane, that will be made of inlier points. In addition, this is made in the presence of a high number of outliers or noisy points that don't belong to the model.

This algorithm has historically been of particular interest in the autonomous driving sector. With it, it is possible to estimate a plane that makes it possible to discern the plane of the road from the rest of the objects that make up a point cloud, known as plane filtering. Therefore, a planar model will be estimated in which the inliers correspond to the road while the outliers are other objects in the scene.

RANSAC receives as input parameters:

- `Point cloud`: set of input points in space.
- `num_coeff`: the number of coefficients indicates the geometric model to be fitted. Three coefficients are needed to fit a 2D line and four for a 3D plane.
- `num_iter`: the number of iterations the algorithm will make to fit the plane. This is a user-defined input.
- `dist_threshold`: maximum distance at which a point can be considered as belonging to the adjusted plane.

To achieve its goal, RANSAC iterates the following steps `num_iter` times:

1. Selects a random number of points belonging to the original point cloud. This set is called *hypothetical inliers*.
2. The model parameters are calculated from the set of hypothetical inliers.
3. The remaining non-model points are tested against the hypothetical model by calculating their distance. If this is less than the threshold, they are added to the model. The hypothetical model is considered good enough if this step involves adding as few points as possible to it.
4. The parameters of the new model are re-estimated after including the new points.
5. The model is evaluated by comparing the relative error, which is quantified according to the number of added inliers.

These series of steps is repeated and the algorithm outputs the model with the lowest error metric, in other words, the one with the fewest number of inliers that had to be added during the previous steps.

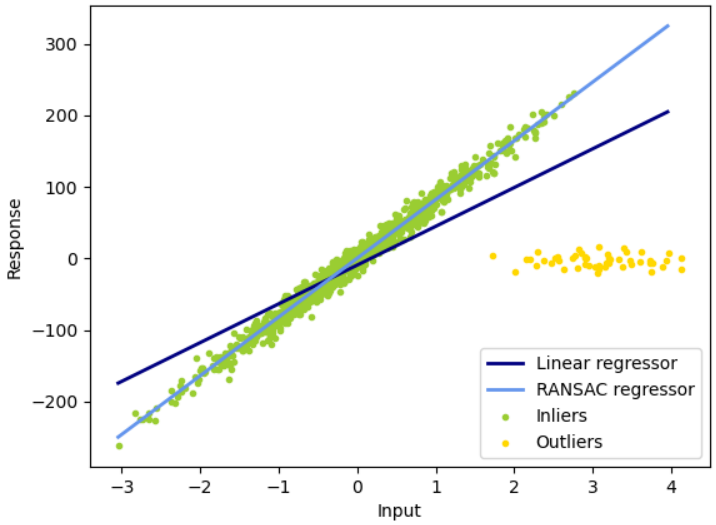


Figure 6.1: RANSAC model fitting algorithm application example
Source. scikit-learn.org. Scikit-Learn Docs: RANSAC Regressor

In conclusion, the great advantage of the RANSAC algorithm is its ability to estimate a geometric model in the presence of a strong presence of aberrant points or noise. It does this by calculating the parameters accurately and being very efficient in terms of computational time. Figure 6.1 shows how the quality of the result increases in the presence of strong outliers that make traditional regressors not powerful enough for such violent environments.

6.2.2 Machine Learning and Clustering Techniques

The natural strength of Machine Learning and unsupervised learning techniques is the extraction of relationships between data sets that can be represented in space. These relationships are also known as features. This is why these types of techniques have been postulated as a viable option when working with three-dimensional point clouds, in which each of the points represents a location dimension in space, to perform object detection tasks. Although other alternatives based on Deep Learning have taken hold in the state of the art, starting down a path using these techniques seems like a good idea.

Clustering techniques belong to unsupervised learning algorithms, which means that you have a set of input data that are not labelled with their corresponding features. The main goal of clustering is to group similar instances together into clusters. The notion of similarity depends on the task at hand: two nearby instances could be considered similar or two instances that belong to the same dense region could also be considered similar. Just like in a classification task, a group of instances with similar features constitute a new set but that in this case no prior information is available.

These techniques have proven useful in a wide range of applications: from customer segmentation in marketing and sales, to anomaly detection when analysing datasets, to applications where they are used for dimensionality reduction. In addition, it has also proven to be viable for the application at hand as it is very useful for contour and shape detection in point clouds.

Clustering algorithms can be classified according to the approach they take to the problem. The most typical classification takes into account two main factors, whether they take into account centroids or density criteria and whether they are hierarchical or not. This results in a matrix classification with four possible combinations.

Based on the criterion of grouping clusters by centroid or density:

- **Centroid-based clustering**: it is considered as one of the most simplest and effective clustering algorithm types. In these algorithms, each cluster is characterized by a centroid point that acts as a seed. Points that are in close locations or proximities to the centroid are assigned to the respective cluster.

  Typically, these methods act iteratively on each point by measuring the distance to the centroid (either by Euclidean, Manhanttan or Minkowski distance) and deciding whether to assign the point to the cluster according to certain predefined user parameters, such as the number of clusters to obtain or the maximum distance between points.

- **Density-based clustering**: these methods take as the main criterion for cluster agglomeration the density of points or data that coincide in a region. Therefore, clusters will be considered as those regions that have a higher density of points and are separated from each other by regions of low density.

  Density-based clustering is one of the prominent paradigms for clustering large data sets in the data mining community. It has been extensively studied and successfully used in many applications. These methods do not require the number of clusters as input parameters, nor do they make assumptions about the underlying density or the variance within the groups that may exist in the data.

Based on the criterion of whether they are hierarchical or flat:

- **Flat clustering**: these methods create a flat set of clusters without any explicit structure or relationship between them. In other words, is where the scientist tells the machine how many categories to cluster the data into.

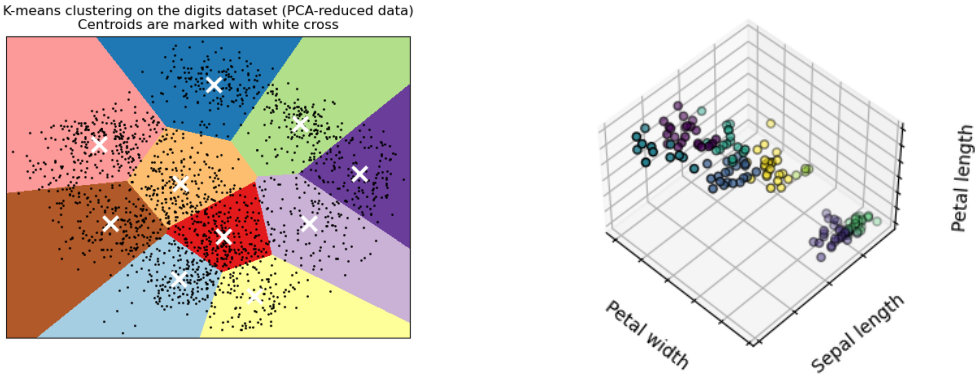
- **Hierarchical clustering**: is a method in which the clustering begins with a top-bottom approach in terms of hierarchy. These algorithms perform a descomposition of the hierarchy on smaller clusters based on pre-defined algorithms. They have a very useful tool for visualisation, the dendogram. Here you can see the relationships between the hierarchy and the different subclusters in order to be able to finetune the parameters.

During the following subsections, two of the most representative clustering techniques will be studied: K-Means [67] and DBSCAN [68].

6.2.2.1 K-Means algorithm

K-Means [67] is a flat centroid-based clustering algorithm that is considered as the main reference in this category and one of the most widely used clustering algorithms. It was proposed by Stuart Lloyd at Bell Labs in 1957, but it was kept confidential until 1982, the release date of [67]. Other authors claim that the main idea was proposed at [69] in 1956.

The algorithm is initialised with a number of clusters to be found as a parameter, a characteristic of the dataset to be clustered that may not be trivial. Each of the instances or points is assigned to each of the clusters, assigning it a label with the cluster number to which it belongs. In addition, the algorithm finds the centroids of all clusters. The vast majority of the points will be associated to their respective clusters in a satisfactory way, but the algorithm does not perform well for clusters with large differences in size and diameter. This algorithm works iteratively. In an initialisation, centroids are randomly positioned and clusters start to form, associating each point to the nearest one. After this first iteration, the cluster centroids are updated by computing the average of the points belonging to the cluster and the process is repeated. The process is repeated until the centroids are positioned in the same position after a number of iterations, i.e. converge. The algorithm itself guarantees convergence, but does not guarantee convergence to the optimal solution. To deal with this, there are centroid initialisation techniques.



(a) Applied over Digits dataset (b) Applied over Iris dataset

Figure 6.2: K-Means clustering algorithm application examples
Source. scikit-learn.org. Scikit-Learn Docs: K-Means

Figure 6.2a shows an application of the algorithm on the Digits dataset. This type of representation is called Voronoi tessellation, in which each cluster is represented by a coloured area and its centroids are marked with white crosses. In Figure 6.2b, another 3D representation on the Iris dataset can be seen, to solve a typical flower species classification task, but in this case, without labels or prior information.

Estimating the optimal number of clusters is also a tricky question. Other disadvantages of the K-Means algorithm are that it takes several iterations to reach a suboptimal solution, it does not separate clusters of different sizes and shapes well, and it is not able to detect anomalies, as no point is left without a cluster. These limitations make it unfeasible for our application.

6.2.2.2 DBSCAN algorithm

DBSCAN [68] is an acronym for *Density-based spatial clustering of applications with noise*. It is the leading exponent of density-based clustering algorithms and to this day continues to be used in some form or another in state-of-the-art applications such as [70].

The algorithm defines all clusters as continuous zones or regions of high point density. It also highlights when these regions are separated by other regions of low density. DBSCAN needs two finetuneable parameters as input:

- ϵ: radial distance from an instance to be explored in search of neighbouring instances.
- `MinPts`: minimum number of instances in an ϵ distance for an instance to be considered a core point.

DBSCAN algorithm consists of:

1. The first step is to visit all points or instances in the dataset. For each point, the number of neighbours within a distance of radius ϵ is noted.
2. Instances are classified. If an instance has a number of neighbours within ϵ greater than or equal to `MinPts`, it is considered a *core instance*. If an instance has a number of neighbouring points greater than 0 but less than `MinPts` and any of the neighbours is a *core point*, it is classified as a *border point*. In case the instance does not have any neighbours, it is considered as a *noise point*.

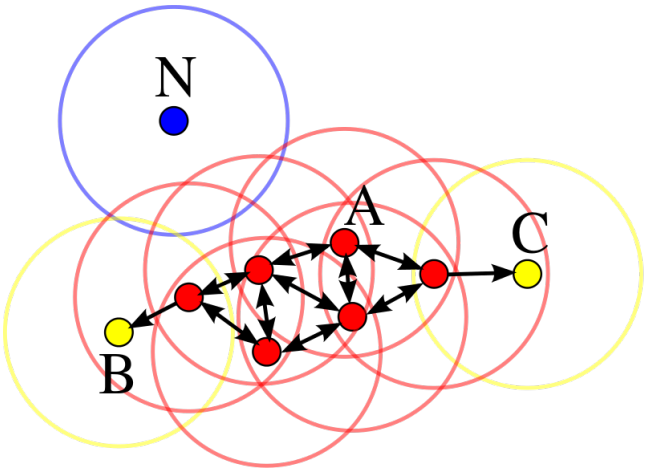


Figure 6.3: DBSCAN instances: core, border and noise points
Source. es.wikipedia.org. DBSCAN

Figure 6.3 shows how the red points (A) are *core points*, since they have in their neighbourhood a number of points greater than `minPts`. For the case of the figure, `minPts` $= 3$ or `minPts` $= 4$ could be applicable. The yellow points (B and C) are *border points* since they have only one point in their neighbourhood that is a *core point*. The blue point (N) has no point in its neighbourhood and is therefore a *noise point*.

3. Clusters are built. Each of the clusters is constructed by grouping all *core points* separated by a distance smaller than ϵ and all their reachable *border points*. The *core points* are called *directly reachable*, while the *border points* are called *indirectly reachable*. *Noise points* are outside the clusters and are considered *anomalies*. In the case of Figure 6.3, the cluster would consist of the red and yellow dots (A, B and C).

In conclusion, it can be said that DBSCAN is a very simple and powerful algorithm that gives satisfactory responses to the problem of finding clusters in datasets, regardless of geometries, sizes or diameters. Moreover, it is robust to outliers because it is able to categorise them as anomalies and leave them out of a cluster, without contributing noise to the system. It only requires two hyperparameters to adjust and is not computationally expensive. Accepting its limitations, when faced with clustering regions of varying density, it is the best (efficient and easy to implement) option available to carry out the task of this work.

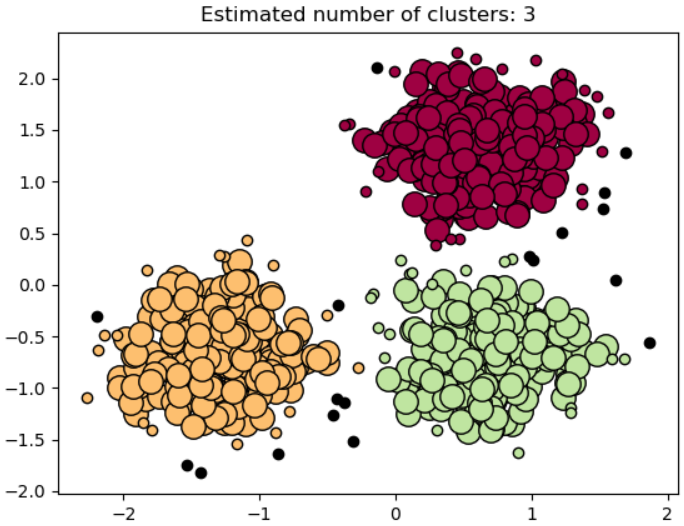


Figure 6.4: DBSCAN clustering algorithm application example
Source. scikit-learn.org. Scikit-Learn Docs: DBSCAN

An example of an application of the algorithm can be seen in Figure 6.4. It can be seen how it separates each of the three point balls and categorises points that do not clearly belong to any of them as anomalies (dark grey points), one of its greatest strengths.

6.2.3 Kalman Filters in Object Tracking

As discussed in Chapter 3, traditional techniques offer high throughput and performance for the task of tracking a vehicle over time by receiving inputs from various sensors. This is why today the Kalman Filter, in its modern, non-linear versions, remains at the forefront of the problem.

The Kalman Filter [11] was devised in the 1960s by the Hungarian-American mathematician Rudolf E. Kalman. The algorithm uses a series of observed measurements over a period

of time, which may be accompanied by noise, inaccuracies and uncertainties, and produces estimates for the future values of the unknown variables. These values tend to be better and more accurate than those based on a single measurement since a probabilistic estimate is made of the possible combinations and distributions that may occur over time.

It has been widely used in various sectors, such as econometrics or navigation, but for the particular case that concerns us, it has been especially relevant for the development of tracking and estimation applications through sensor fusion. It is ideal for real-time systems whose variables are continuously changing and has been considered one of the "most useful mathematical inventions of the 20th century".

6.2.3.1 Linear Kalman Filter

The Kalman Filter is a recursive estimator algorithm. This means that only the estimate of the previous state and a current measurement are needed to obtain the estimate of the current state. Therefore, this recursive algorithm will always consist of a succession of prediction and update stages, with an initialisation stage to make the first estimate.

The input to a Kalman Filter is the description of a system in state variables. A matrix $\vec{x}$ is responsible for providing a faithful dynamic representation of the system. For illustrative purposes, an example will be used in which a robot will be modelled using the state variables position and velocity. In matrix notation, it would look like this.

$$\vec{x} = \begin{pmatrix} position \\ velocity \end{pmatrix} \tag{6.1}$$

As each of the system outputs will be estimated from a possible combination, each of the state variables will be considered as a Gaussian and random variable and will therefore be defined by a mean (μ), which will indicate the centre of this Gaussian, and a variance (σ^2), which models the uncertainty of the measurement. This will make some areas more probable than others, according to criteria of precision and confidence in the measurement. Figure 6.5 shows an example of these two measures for the two-dimensional plane formed by the state variables in this example.

By also looking at Figure 6.5, information can be extracted on the relationship between the state variables. In the case at hand, both variables are uncorrelated. However, to store the degree of relationship between each of the variables, the covariance matrix appears. This matrix will have a size $n \times n$ where n is the number of initial state variables. Each of the elements Σ_{ij} of the matrix denotes the relationship of the i-th variable to the j-th variable. The covariance matrix is often referred to as Σ.

If in our example, the position variable is taken as p and the velocity variable as v, the covariance matrix will be:

$$\Sigma = \begin{pmatrix} \Sigma_{pp} & \Sigma_{pv} \\ \Sigma_{vp} & \Sigma_{vv} \end{pmatrix} \tag{6.2}$$

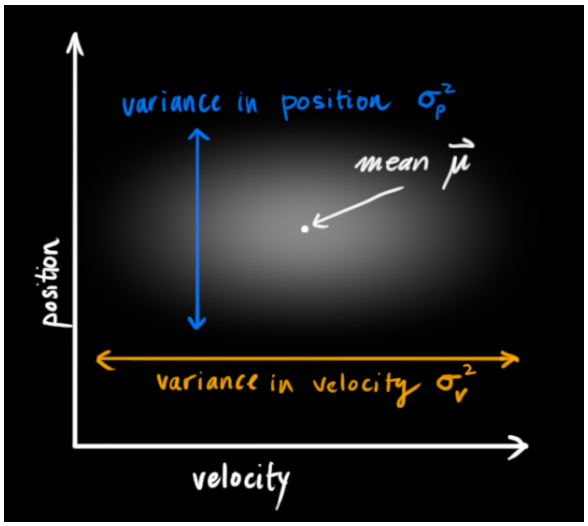


Figure 6.5: Mean and variance for state variables in Kalman Filter
Source. bzarg.com. How a Kalman Filter Works

For each time instant $k-1$, two pieces of information will be used to predict the value of the next instant k: the estimation matrix $\hat{x}_{k-1}$, where the means of the variables are stored, and the aforementioned covariance matrix P_{k-1}. For the example, taking basic equations for a constant velocity motion, a predition matrix can be defined as:

$$F_k = \begin{pmatrix} 1 & \Delta_t \\ 0 & 1 \end{pmatrix} \tag{6.3}$$

The following state may be estimated as:

$$\begin{aligned} \hat{x}_k &= F_k \cdot \hat{x}_{k-1} \\ P_k &= F_k \cdot P_{k-1} \cdot F_k^T \end{aligned} \tag{6.4}$$

If it is desired to expand the Filter to take into account external influences, such as in this example the effect of an expected acceleration on the mechanical elements of the robot rubbing against the ground, it could be stored in a vector $\vec{u}_k$, whose associated gains are stored in the matrix B_k. These elements are called control vector and control matrix, respectively.

In addition, the system may be exposed to uncertainties coming from the limitations of our robot's measurement systems. This will cause an associated term to appear in the covariance matrix and will be added to the prediction of the next state. These uncontrolled influences are called noise Q_k and are added to the initial covariance.

Now, it is possible to have a Kalman Filter capable of combining the dynamic effects of a system and the unwanted external influences in order to predict the next states of the system over time. These expanded expressions are as follows:

$$\begin{aligned} \hat{x}_k &= F_k \cdot \hat{x}_{k-1} + B_k \cdot \vec{u}_k \\ P_k &= F_k \cdot P_{k-1} \cdot F_k^T + Q_k \end{aligned} \tag{6.5}$$

Once the prediction phase is complete, it is necessary to look at the information coming from the sensors to be able to interpret the information of interest that is given about the world, which if recall was being modelled by the state variables. This information will be modelled in the H_k matrix and represent what the user expects to find in the sensor readings. Analogous to what was had in the prediction, the expected magnitudes for the means and covariances of the state variables will be:

$$\begin{aligned} \hat{\mu}_{expected} &= H_k \cdot \hat{x}_k \\ \Sigma_{expected} &= H_k \cdot P_k \cdot H_k^T \end{aligned} \tag{6.6}$$

However, for each of the sensor readings, our system will have a certain probability of evolving to a particular state and may not be the same as what was previously expected. This will also be modelled by a Gaussian of variance R_k and mean $\vec{z}_k$ since there will be a number of states towards which the system will be more likely to evolve than others. In this scenario it is necessary to face the reality that there are two Gaussian surfaces that model the next state the system will be in: the one predicted in the previous step and the one from the sensor readings. To obtain the new state, the ideal is to combine both Gaussians and this can be seen graphically in the two-dimensional plane mentioned at the beginning of this explanation: stay with the area of union of both, the junction.

Predicted measurement:

$$(\mu_0, \Sigma_0) = (H_k \cdot \hat{x}_k, H_k \cdot P_k \cdot H_k^T) \tag{6.7}$$

Observed measurement:

$$(\mu_1, \Sigma_1) = (\vec{z}_k, R_k) \tag{6.8}$$

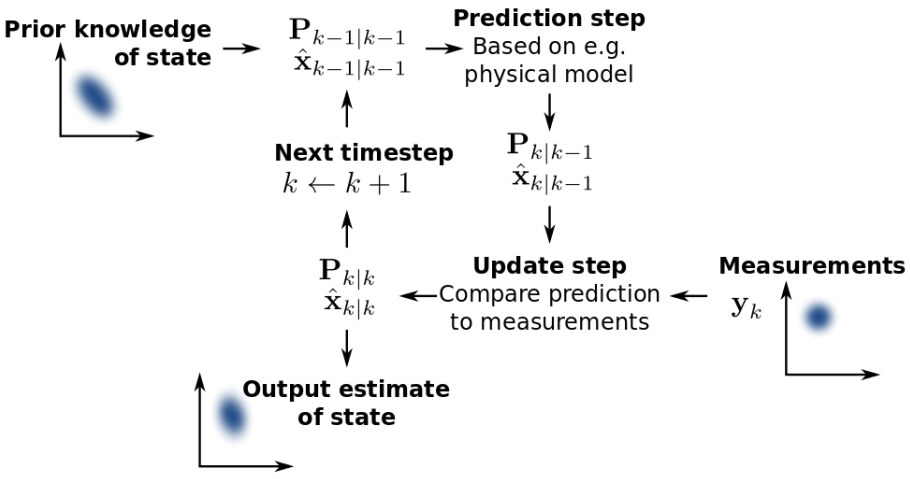


Figure 6.6: Kalman Filter diagram
Source. en.wikipedia.org. Kalman Filter

The overlap between both Gaussians is obtained as:

$$\begin{aligned} H_k \cdot \hat{x'}_k &= H_k \hat{x}_k + K \cdot (\vec{z}_k - H_k \hat{x}_k) \\ H_k \cdot P'_k \cdot H_k^T &= H_k P_k H_k^T - K \cdot H_k P_k H_k^T \end{aligned} \tag{6.9}$$

Where K is the Kalman Gain and is defined as:

$$K = H_k P_k H_k^T \cdot (H_k P_k H_k^T + R_k)^{-1} \tag{6.10}$$

The following status is obtained and the update phase is completed, resulting in:

$$\begin{aligned} \hat{x'}_k &= \hat{x}_k + K'(\vec{z}_k - H_k \hat{x}_k) \\ P'_k &= P_k - K' H_k P_k \end{aligned} \tag{6.11}$$

$$K' = P_k H_k^T \cdot (H_k P_k H_k^T + R_k)^{-1} \tag{6.12}$$

And with this it is possible to estimate the new state of the system. Given the recursive nature of the algorithm, these prediction and update steps are repeated over time in a sequential fashion, feeding in new inputs to the system. For clarity on this explanation, the diagram in Figure 6.6 is attached.

6.2.3.2 Hungarian Algorithm

Hungarian Algorithm is a combinatorial optimisation method that solves assignment problems and allows the combination that minimises a target variable. It was devised in 1995 by the mathematician Harold Kuhn and owes its name to the fact that the foundations of the algorithm are based on the work of two mathematicians of Hungarian origin: Denes Konig and Jeno Egervary. For pedagogical purposes, the Hungarian Algorithm is usually explained by means of an example in which a table is used as data in which the rows and columns are a series of employees of a company and a series of tasks to be carried out, respectively. Each of the cells contains the price that it will take each employee to perform the task. The Hungarian Algorithm allows to find the solution in which the cost to perform all tasks is the lowest possible.

In the case of object tracking in the field of autonomous driving, the algorithm is used to match the detections made at one point in time with those made at a later point in time. It is the tool that makes it possible to recognise the same object over time and to track it.

The algorithm takes 4 steps to solve the assignment problem and receives the cost matrix C as input:

1. **Subtract the minimum of each row from all the elements in the row.** For each row, the minimum element is found and subtracted from each and every element in each row.

$$C'_{ij} = C_i - \min(C_i) \quad \forall \quad j \tag{6.13}$$

2. **Subtract the minimum of each column from all the elements in the column.** In the same way as in the previous step, for the new cost matrix C', find the minimum element of each column and subtract it from all the elements of the column.

$$C''_{ij} = C'_j - \min(C_j) \quad \forall \quad i \tag{6.14}$$

3. **Cover the zeros with a minimum number of rows.** The zeros must be covered with horizontal or vertical lines, which means taking rows or columns and the minimum number of them. If the number of lines (rows or columns) is less than the size of the matrix, continue. If it is greater or equal, the optimal resource allocation has been found.

4. **Create additional zeros.** If the size of the matrix is greater than the minimum number of rows, steps 1 and 2 must be repeated to find additional zeros to arrive at the optimal allocation in step 3.

6.2.3.3 SORT

SORT [71] is an algorithm proposed by Bewley *et al.* in 2016 for tracking multiple objects whose estimates and data associations are made using classical techniques. It was originally based on 2D camera detections. Its name is an acronym for *a simple Online and Realtime Tracking*. At its launch, it was ranked Top 1 in MOT Benchmark for the Open Source Trackers category achieving the highest frequency and accuracy in MOTA metrics. Moreover, it complies with the philosophy of all the techniques that have been used in this work, classical techniques that offer good results to obtain the objective of contextualisation and introduction of the author in the development of perception systems for autonomous vehicles. Within the categorisation made in Chapter 3, the SORT algorithm is considered to belong to the tracking-by-detection category.

The algorithm makes use of a motion model to propagate the identity of a detected object from one timestamp to the next. The identity of each detected object is stored in a vector, which for the original version of the algorithm consisted of 7 variables, but for the implementation in this work it will be 9 variables to store the information of the spatial location in the 3D world and the velocity from the RADAR.

$$X = \begin{bmatrix} x & y & s & r & \theta & x' & y' & s' & \theta' \end{bmatrix}' \tag{6.15}$$

Where x and y represent the spatial location of the object, and although in the original version of the SORT algorithm they indicate an area as seen from a 2D camera, in this implementation it is taken as a bird's eye view position that eliminates height. Following the equation, s represents the scale of the object and r its aspect ratio. The angle θ indicates the orientation of the object in space. In addition to these 5 observable variables, 4 inferred variables are added. The first two x' and y' are fed by the RADAR data, while the rest are predicted by the Kalman Filter.

When a detection is associated to an object in the scene, a Bounding Box is produced and used to update the state of the object and predict its behaviour at the next timestamp via Kalman Filter. If there is no detection associated with the object, it is predicted without correction or update using the motion model.

To associate the detections to the objects stored by the algorithm, a cost matrix is proposed that computes the Intersection over Union of all detections with the objects. This provides the necessary input to apply the Hungarian Algorithm and see which new detection corresponds to the saved object, an assignment problem.

The identities of the objects and their motion models are created as they appear in the scene and are destroyed when they no longer appear in the detections. In addition, the algorithm has two user-customisable parameters that allow defining the behaviour in case of object creation and destruction. As a mechanism to avoid false positives, the user can decide the number of times an object is detected to be considered by the algorithm. In other words, if the same target appears n times in the detections, its identity and its motion model are created so that it counts in the processing. The second parameter is the number of times an object is not detected so that its identity is destroyed and it is not counted.

6.2.4 System development

The architecture proposed for this project consists of parallel processing of data from LiDAR and RADAR sensors that are generated in the CARLA simulator (described at Tables 6.1 and 6.2, respectively) and sent to the ROS framework through the CARLA ROS Bridge tool. Once the data is received, it is processed. This will implement traditional and efficient techniques that continue to appear in state-of-the-art proposals, such as plane segmentation using RANSAC or the DBSCAN clustering algorithm, and allow 3D Object Detection by means of an analysis of the vehicle's surroundings. With the detections and a sensory fusion between data from both sensors, Multi-Object Tracking is carried out, which will allow information to be obtained over time from the monitored road entities.

For the project to be considered as completed, the following objectives must be met.

- Implementation of a LiDAR sensor data processing pipeline using the `Open3D` framework for Object Detection.
- Implementation of a RADAR sensor data processing pipeline using the `Scikit-Learn` framework for Object Detection.

Attribute	Type	Default	Description
channels	int	32	Number of lasers
range	float	100	Maximum distance to measure or raycast in meters.
horizontal_fov	float	30.0	Horizontal field of view in degrees.
upper_fov	float	10.0	Angles in degrees of the highest laser.
lower_fov	float	-30.0	Angles in degrees of the lowest laser.
rotation_frequency	float	10.0	LiDAR frequency rotation.
points_per_second	float	56000	Points generated by all lasers per second.

Table 6.1: Main parameters of LiDAR sensor in CARLA
Source. carla.readthedocs.io. LiDAR Sensor

Attribute	Type	Default	Description
horizontal_fov	float	30.0	Horizontal field of view in degrees.
vertical_fov	float	30.0	Vertical field of view in degrees.
points_per_second	float	1500	Points generated by all lasers per second.
range	float	100	Maximum distance to measure or raycast in meters.

Table 6.2: Main parameters of RADAR sensor in CARLA
Source. carla.readthedocs.io. RADAR Sensor

- Development of a sensor fusion mechanism between LiDAR and RADAR detections to obtain maximum information from the environment.
- Development of a Multiple Object Tracking system applying a SORT-based algorithm, whose inputs will be the objects detected in the previous pipelines.
- Integration of the complete system into the ROS software development framework for robots.

6.2.4.1 File Hierarchy and Modules

The repository of this project follows the typical structure of a ROS project written in Python. During the development of the project, attention has been paid to the community standard for repository creation and maintenance. The directories that make up this project are:

- launch: launch files are stored in this directory. These are ROS-specific files that allow to initialise a set of framework elements together: nodes, parameters, plugins and tools. For this work, launch files are designed to launch the simulation and the processing pipeline separately.

- `rviz`: this directory stores the configuration files of the RViz simulator, a tool that helps the visual debugging of the application by means of an interface where the messages flowing through the ROS topics can be observed.
- `src` this directory includes the Python script (`main.py`) in charge of the complete running of the project.
- `src/modules`: this directory stores Python scripts that act as modules and encapsulate entities and functions that serve the main script for the running of the project. These modules are: `objects.py`, `detection_helper.py` and `types_helper.py`.

6.2.4.2 Setting up ROS infrastructure

To start with the development of the application, first a ROS-based infrastructure is set up so that all messages coming from LiDAR and RADAR can be processed in a parallel and synchronised way. For this purpose, in the main script we make use of a class that will contain the initialisation of the node that will manage the application and all the publishers and subscribers through which the ROS messages will circulate are defined.

As synchronisation elements, use will be made of the `tf` and `messages_filters` libraries. The first one provides a framework of transformations between the reference systems of both sensors, since the LiDAR is positioned on the roof rack of the car, while the RADAR is positioned on the front. The second one provides ROS with a software synchronisation interface of the messages flowing through the topics. For the realisation of this work, it is of special interest to process messages from both sensors synchronised in time, so that the fusion can be carried out successfully.

To finish this step of the work, launch files are created to automate the launching of the simulation and the algorithm, and the display options of the RViz simulator are customised. A hierarchical diagram of the execution of the elements present in the repository is shown in Figure 6.7.

When the user executes the launch files, the system will launch the main script, the associated modules and the simulator with the appropriate configuration for this job.

6.2.4.3 LiDAR data Processing Pipeline for Object Detection

LiDAR sensor data are obtained via CARLA ROS Bridge. They are received as `PointCloud2` type ROS messages, in which the XYZI information of the point cloud is encoded: spatial location and reflection intensity. For the purposes of this work, the latter fourth coordinate is ignored and only spatial data is used. These fields can be seen in a detailed way in Table 6.3.

For the processing of LiDAR data, a pipeline is proposed based on the traditional Machine Learning techniques explained in this Chapter for 3D Object Detection. At source, the point cloud provided by the system provides a 360° view around the vehicle. However, as one of the main objectives is the fusion of LiDAR data with RADAR data, and this second sensor is

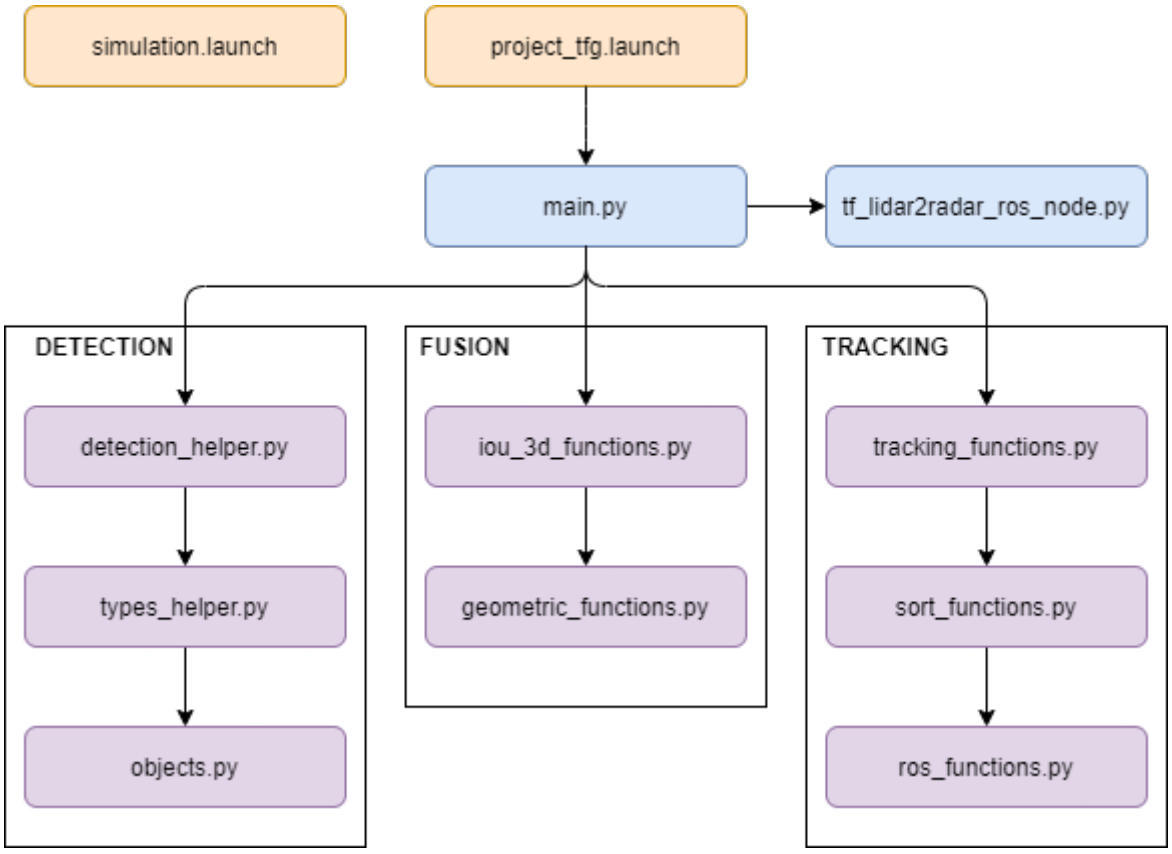


Figure 6.7: Hierarchy diagram of Detection and Multi-Object Tracking system
Source. Own elaboration.

placed at the front of the car, an angled filtering of the cloud is performed, limiting the field of view to the front 80° of the vehicle.

After having eliminated part of the point cloud, it is proceeded to the first technique belonging to the pipeline, whose steps are implemented by means of the `Open3D` library. Applying the RANSAC algorithm, it is proceeded to a segmentation of the road plane that eliminates a large part of the point cloud that is not of interest for the analysis. As previously explained, this algorithm takes as input a number n which is fixed at 3 to determine that the geometric body to be fitted is a 3D plane. In addition, it takes as parameters the number of iterations to be performed and the limit distance between points to belong to the plane. These parameters will be further refined to improve the results. The result of the plane segmentation consists of the plane-fitting cloud (inliers) and its complementary (outliers). Although the outliers cloud will be used for further processing, both clouds are published as intermediate results via ROS messages.

The next step is the application of the DBSCAN unsupervised clustering algorithm. As explained in previous chapters, it is a technique that continues to be very useful in the state of the art. The algorithm receives as inputs the cloud of points to be clustered, the minimum number of neighbouring points for a point to be considered core and the epsilon distance at which the above parameter is evaluated. As output, it provides the user with a list of labels whose length is equal to the number of points in the cloud. A point that has been labelled as anomalous and does not belong to any cluster will have a label -1, while the rest of the points

Attribute	Type	Description
`x`	`float32`	X-coordinate in the Cartesian location space.
`y`	`float32`	Y-coordinate in the Cartesian location space.
`z`	`float32`	Z-coordinate in the Cartesian location space.
`Intensity`	`float32`	Intensity of the reflection on the target.

Table 6.3: Point fields of LiDAR data from CARLA ROS Bridge Source. `github.com`. `lidar.py` script to handle LiDAR sensor.

will be numbered from 0 to N, N being the total number of clusters in the scene. With this list and the point cloud, custom Cluster objects are constructed to represent each of the clusters and their most important attributes: points, centre and velocity (if the cluster has this data associated with it, as will be the case in the RADAR). These objects have methods associated with them that allow conversion to ROS messages of the `visualization_msgs::Marker` type and to other objects of the Bounding Box type.

It didn't take many iterations to realise that such a violent and diverse environment as Autonomous Driving situations make the performance of Machine Learning models highly dependent on the initialisation of their parameters. It is observed that for the same target (especially in vehicles) several clusters appear for the same target, which makes the program understand that several objects are located in the same space, whereas in reality it is only one. This led to additional processing steps being added in which certain clusters were merged if the Euclidean distance between them was less than a certain user-defined limit. The algorithm does the following:

1. A list of length equal to the number of clusters found at that time instant is generated, indicating whether these clusters have been processed.
2. The list of such clusters is iterated over:
   - If the cluster has already been processed, it moves on to the next cluster.
   - On the other hand, if it has not been processed yet, it is iterated over the rest of the clusters and if any of them has a distance between centres smaller than the defined limit, they are merged.
3. The result of this search is added to a new list: if there has been no merge, the original cluster is added, while if there is, they are added as a single new cluster. This list is returned.

This algorithm makes it possible to merge clusters that are more difficult for DBSCAN to process and makes the entire program less dependent on the initialisation of its parameters.

Once the final clusters are in place, custom Bounding Box objects are created. These objects allow a quick conversion between Cluster and Bounding Box and in their initialisation a series of parameters and attributes are calculated that allow the spatial definition and the main characteristics of the object that both represent. They also have methods for display in RViz in

the form of `jsk_recognition_msgs::BoundingBox` messages, whose information fields are detailed in the Table 6.4. And this step ends the processing of LiDAR data for 3D Object Detection.

Attribute	Type	Description
`header`	`Header`	Standard metadata for higher-level stamped data types.
`pose`	`Pose`	A representation of pose in free space, composed of position and orientation.
`dimensions`	`Vector3`	This represents a vector in free space (x, y, z).
`value`	`float32`	Placeholder for a numerical value (used for speed in RADAR data).
`label`	`uint32`	Placeholder for a label value (used for ID in tracking tasks).

Table 6.4: Elements of a `jsk_recognition_msgs::BoundingBox` message Source. `docs.ros.org`. ROS Documentation for `jsk_recognition_msgs::BoundingBox` message.

6.2.4.4 RADAR data Processing Pipeline for Object Detection

RADAR data are also obtained from the CARLA ROS Bridge tool. As with LiDAR, the data is translated and sent to ROS via a `PointCloud2` message type. In this case, the RADAR point cloud is more complex, as it supports two ways of working with this sensor. Normally, if one were working in the real world with physical sensors, the sensor measurements provide information that, using digital signal processing mechanisms, is easily translated into cylindrical coordinates in space and Doppler velocity. In this work, as we are working with simulation data, it is chosen to acquire the data in Cartesian coordinates for a direct translation to the LiDAR reference frame (which will act as the main sensor). In addition, the Doppler velocity information of the targets is taken. Table 6.2 shows all the parameters that are encoded in the RADAR point cloud provided by CARLA.

Attribute	Type	Description
`x`	`float32`	X-coordinate in the Cartesian location space.
`y`	`float32`	Y-coordinate in the Cartesian location space.
`z`	`float32`	Z-coordinate in the Cartesian location space.
`Range`	`float32`	Distance to target in cylindrical location space.
`Velocity`	`float32`	Doppler velocity relative to vehicle speed.
`AzimuthAngle`	`float32`	Angle with respect to XY axes in cylindrical location space.
`ElevationAngle`	`float32`	Angle with respect to XZ axes in cylindrical location space.

Table 6.5: Point fields of RADAR data from CARLA ROS Bridge Source. `github.com`. `radar.py` script to handle RADAR sensor.

Therefore, the first step to be taken in this pipeline is the extraction of the information fields of interest for this work. According to the table above, the spatial information for a Cartesian workspace is defined in the first three columns, while the Doppler velocity is encoded in the

fifth column. This means that instead of working with a 7-dimensional point cloud, we will work with a 4-dimensional XYZV point cloud.

At this point, it is proceeded to use the `Scikit-Learn` framework to process this point cloud. As it is observed that the RADAR point cloud has much fewer points than the LiDAR one, it is not considered appropriate to make a segmentation of the plane using the RANSAC algorithm, since these are already sufficiently spaced so that the clustering algorithm can define and separate areas of different density when obtaining the objects in the scene.

The DBSCAN algorithm is applied as a clustering technique to extract those groups of points that belong to an object. This is followed by the extraction of Bounding Boxes. This process, even with small differences, is quite similar to the LiDAR processing pipeline. The main difference is that each and every one of the objects obtained from this point cloud has the relative velocity or Doppler velocity information encoded, which is obtained by performing an arithmetic mean with aberrant filtering for all the points belonging to the cluster.

6.2.4.5 Fusing LiDAR and RADAR data

One of the ideas that has been repeated throughout the development of this work is that sensing systems become more robust as the amount of information received by the sensors increases. Having several sensors, a sensory fusion between LiDAR and RADAR data will be performed based on an Intersection over Union 3D algorithm that will obtain as a result Bounding Boxes that will represent the objects in the scene whose position will be given by the position of its counterpart LiDAR detection, and its velocity will come from its RADAR detection. This decision is made because the LiDAR point cloud is denser and can be better acted upon by DBSCAN, whereas speed information can only be obtained from the RADAR point cloud.

For this purpose, an algorithm is run that runs through a list of objects detected by the LiDAR. In a nested way, a list including the objects detected by the RADAR is traversed. For each object (LiDAR detection and RADAR detection) the 3D IoU is calculated. If this is greater than a certain user adjustable threshold, then both detections can be said to belong to the same object and are merged. The result of the fusion is entered into a list that will serve as input to the Kalman Filter for Tracking.

For the construction of the 3D boxes with which Intersection over Union is made, the information of the Bounding Box is taken as input and the 8 corners of it are obtained. Both boxes are taken to a latent space in which they are referenced to the same frame and polygon intersection and union operations are performed. This work is supported by the use of the `scipy` module.

Thanks to this process, a late fusion of the objects is achieved, since the fusion is generated from the detections of both sensors (see Chapter 3), which are working in a complementary way. The result of this fusion is published in a ROS topic as a message of type `visualization_msgs::MarkerArray`.

6.2.4.6 Tracking objects with SORT

Once the Object Detection and Sensor Fusion stages are over, it is time to temporarily track True Positive Detections. For this purpose, an algorithm based on **SORT** [71]: *a simple Online and Realtime Tracking* is implemented. It is a technique that allows a tracking of multiple objects whose provenance is given by rudimentary data associations.

In order to adapt the SORT algorithm to the proposed architecture, several of its concepts need to be modified, but its working principles and simple philosophy are maintained. First, the motion model vector is adjusted to track from a bird's eye view perspective, which eliminates the height dimension in the Cartesian plane. This way, the 2D philosophy of the camera can be adapted to the 3D world.

With this new motion model vector, it is proceeded to define the transition function F, the covariance matrix and the way to compute the error. For the association stage using the Hungarian Algorithm, an Intersection over Union can be computed in bird's eye view that will compose the elements of the matrix to assign a detection in the past frame with a new detection.

Finally, the algorithm is adjusted so that 3 detections in a row are necessary for its identity and motion model to be created. In addition, if the target is lost from sight, its identity is deleted and if it reappears it will be tracked with another identity.

6.2.5 Experimental Results

During the following section, a qualitative description of the results of the system and its behaviour in situations that are considered scenarios of interest for autonomous driving is given. To conclude this section, an analysis of the time taken for each of the processing phases of the system, as well as whether it is possible to be executed in real time, is attached.

LiDAR Object Detection evaluation

First, the performance of the LiDAR Object Detection sensor that is performed on the point cloud provided by the CARLA simulator will be evaluated. For this purpose, some images taken from the RViz simulator are presented in which the camera monitoring the scene can be seen in the lower left part, while the right part of the figures shows the sensor point cloud and the detections in the form of cubic and cyan-coloured Bounding Boxes.

As discussed in this chapter, the point cloud is first filtered to focus attention on the front of the vehicle. This phenomenon can be observed in Figure 6.8, where the original point cloud provided by the simulator can be seen in purple and the filtered part corresponding to the frontal 80° can be seen in white.

Figure 6.8 shows a scenario in which a Bounding Box is extracted for a vehicle parked on the side of the road while the ego-vehicle passes it. The shape of the Bounding Box is an elongated

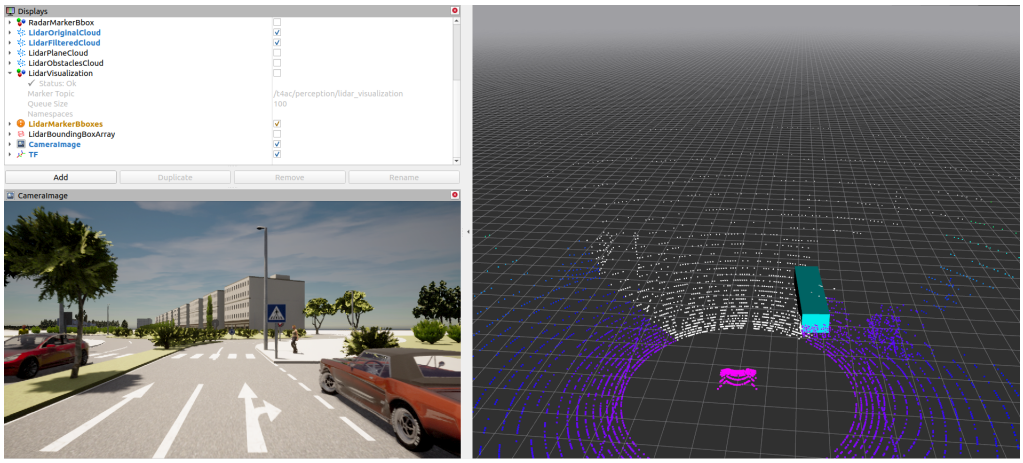


Figure 6.8: LiDAR Object Detection for passing vehicles parked on the side
Source. Own elaboration.

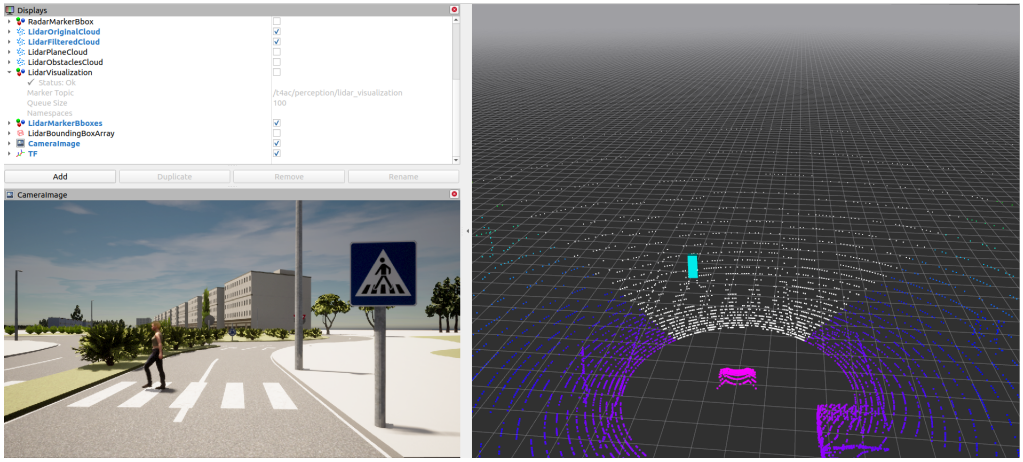


Figure 6.9: LiDAR Object Detection for dynamic pedestrian on a zebra crossing
Source. Own elaboration.

rectangle whose length and height correspond to the overall dimensions of the detected vehicle while the width is only that which falls within the range of the front point cloud.

In the same scenario and a few steps further on, a situation can be observed in Figure 6.9 where the ego vehicle stops at a zebra crossing that is being crossed by a moving pedestrian. In this case, the pedestrian is completely within the frontal point cloud of the vehicle. Consequently, the detector is able to cluster the pedestrian's silhouette and generate a Bounding Box whose dimensions are equal to those of the pedestrian in width, height and length.

The following scenario shows a situation where the ego-vehicle is waiting to start at a roundabout. In the meantime, a car is approaching him from the left. The frame shown in Figure 6.10 shows the moment when the target car is in front of him and it is observed how the detector is able to identify the vehicle as an obstacle resulting in a large cyan-coloured Bounding Box.

The last situation in which the behaviour of the object detector for high-level LiDAR is evaluated is shown in Figure 6.11. The scenario is similar to the first one shown, with a vehicle parked on the side of the ego but in this case the distance to the vehicle is much larger than in

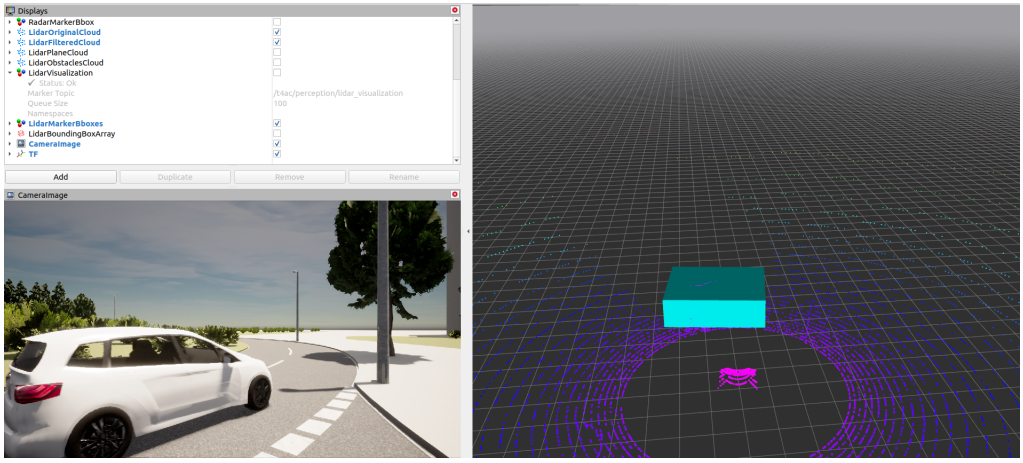


Figure 6.10: LiDAR Object Detection for a vehicle in front of ego at a roundabout Source. Own elaboration.

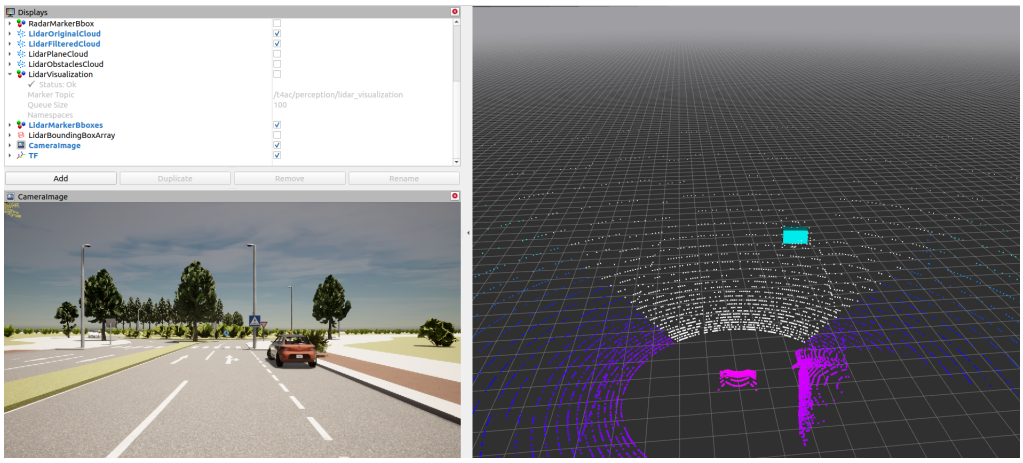


Figure 6.11: LiDAR Object Detection for stationary vehicle parked on the side Source. Own elaboration.

the previous case. The Bounding Box obtained in this case is much smaller and only the rear of the vehicle is detected as a target.

Therefore, it can be concluded that the object detector based on DBSCAN and RANSAC is able to fulfil its functions in various scenarios. The height at which the sensor is positioned on the car roof rack and the richness of the cloud points allow objects to be easily detected and clustered in controlled environments.

RADAR Object Detection evaluation

Next, it is time to evaluate the high-level behaviour of the DBSCAN-based object detector for the RADAR sensor. As with its counterpart LiDAR sensor, three frames obtained from simulated scenarios in the CARLA simulator are shown in which it is subjected to different typical driving situations.

The first thing to notice in Figure 6.12 is that the RADAR point cloud is less dense than the LiDAR one, as it has fewer points. This is a quality of the CARLA simulator that simulates the

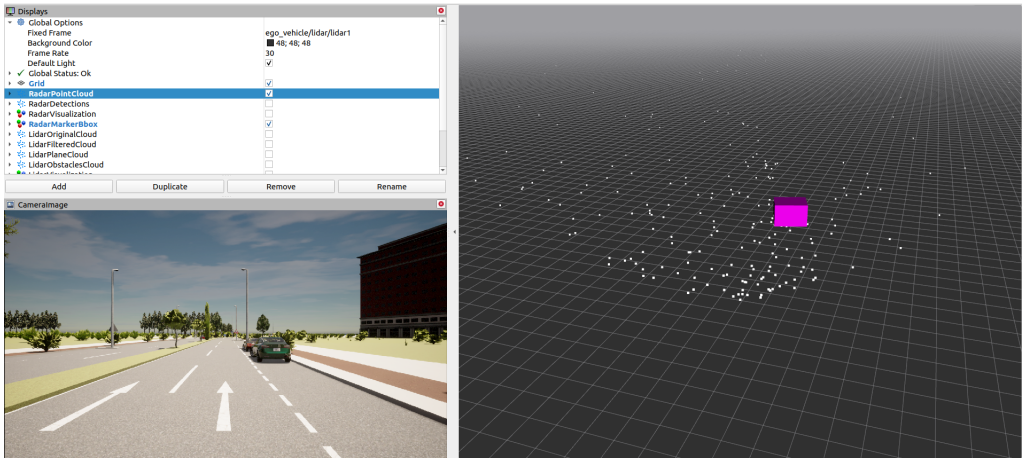


Figure 6.12: RADAR Object Detection for stationary vehicle parked on the side
Source. Own elaboration.

performance of the sensor in real life, in which this sensor obtains fewer detections and less detail of the scene. Another detail to take into account is the position in which this sensor is placed. As it is placed at the front of the vehicle, it is at a shorter distance from the ground and therefore the result of the Bounding Boxes (which for RADAR are coloured in magenta) will have different dimensions.

The first scenario in which RADAR is assessed is exactly the same as the last one in which LiDAR was assessed. In this case, a vehicle is parked on the side of the road approximately 10 metres from the ego vehicle. In this case, the Bounding Box generated on the rear of the vehicle is smaller than the LiDAR but the vehicle is detected satisfactorily.

The second scenario in Figure 6.13 shows the vehicle stopped at a zebra crossing with a moving pedestrian. This case was also evaluated in LiDAR and it can be seen that the RADAR Bounding Box is less high than its counterpart and does not take into account the full height of the pedestrian. Even so, it is able to detect it correctly. This is due to the position of the sensor in the vehicle, which means that the points in the point cloud do not reach the whole pedestrian due to the vertical field of view.

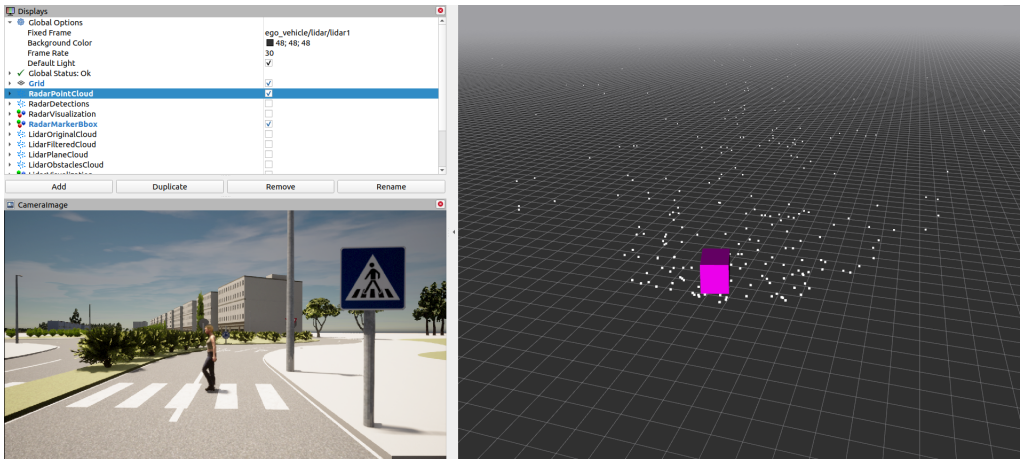


Figure 6.13: RADAR Object Detection for dynamic pedestrian on a zebra crossing
Source. Own elaboration.

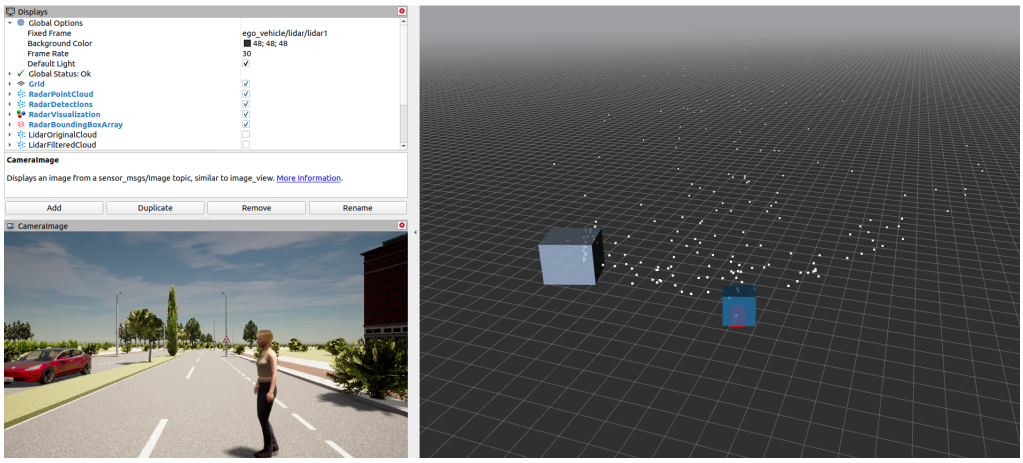


Figure 6.14: RADAR Object Detection for passing vehicles parked on the side
Source. Own elaboration.

The third figure (6.14) shows the qualitative behaviour of the pipeline in a somewhat more adverse scenario, in this case, a pedestrian is simulated crossing the road inappropriately and standing still in the middle of the road. In addition, other vehicles act as actors in the scene. In this frame, it can be seen how the pipeline is able to successfully obtain blue-colored Bounding Boxes for both the stationary pedestrian and the vehicle approaching from the opposite lane of the road.

Sensor Fusion evaluation

Therefore, it can be seen that both sensors have the ability to estimate the position of the objects of interest, although it can be observed that the LiDAR takes better Bounding Boxes thanks to the position it has on the vehicle. This reinforces the idea that the result of the fusion of both sensors is the position estimated by the LiDAR with the velocity information obtained with the RADAR. With this analysis of the high-level behaviour of the Object Detection system for RADAR, we are in a position to fuse the objects from both sensors.

In the case of Sensor Fusion, the Bounding Boxes generated by LiDAR and RADAR detections will be perceived as overlapping in space. Thus the Intersection over Union algorithm will be able to match both detections. The result of the fusion is the location from the LiDAR detection and the velocity from the RADAR detection. The first scenario is shown in Figure 6.15. It shows a vehicle parked on the left side of the road. In two different colours, LiDAR and RADAR Bounding Boxes can be seen. Thanks to this, the algorithm is able to complete the sensory fusion for this scenario.

A few metres further on, but in the same scenario, the scene contained in Figure 6.16 occurs. In this case, the ego vehicle had stopped in front of the roundabout and a vehicle was approaching from its left. Once the vehicle has resumed driving, LiDAR and RADAR detections can be seen to occur for a target being driven after. In addition, not being on a straight road, there is a change of orientation that makes the scenario even more difficult, but which the implemented architecture solves satisfactorily.

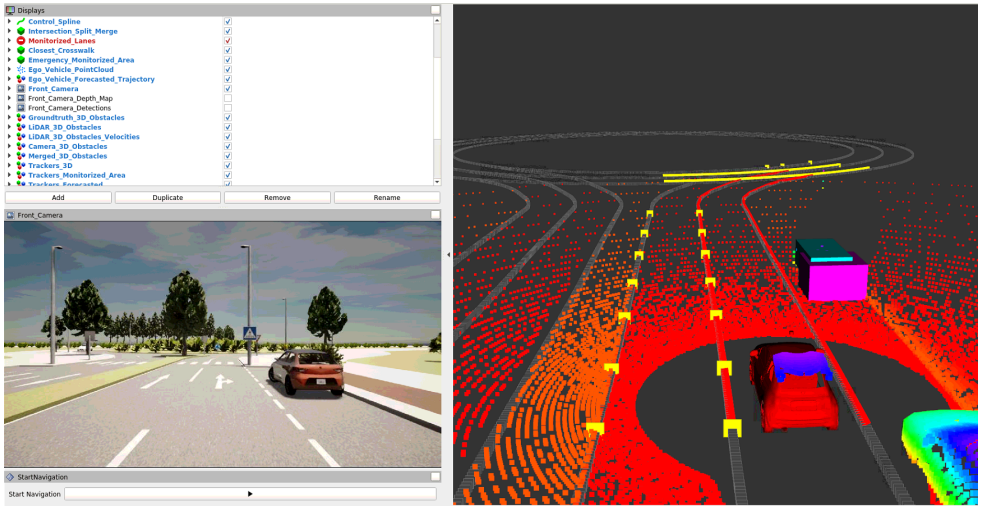


Figure 6.15: Sensor Fusion for stationary vehicle parked on the side
Source. Own elaboration.

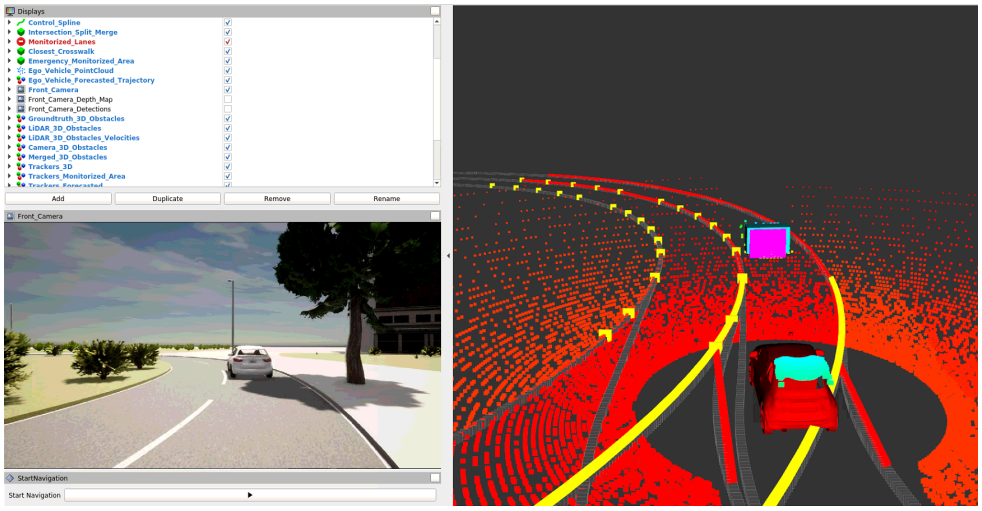


Figure 6.16: Sensor Fusion for dynamic vehicle pursuit at roundabout
Source. Own elaboration.

Changing the scenario, Figure 6.17 shows the simplest case where a vehicle is being pursued on a straight road. As it had solved the previous case, this one was expected to be solved as shown.

Therefore, both Object Detection pipelines have converged in this Late Sensor Fusion and a diverse set of simple and controlled scenarios can be solved. It should be noted that a possible extrapolation to other scenarios would require a fine tuning of parameters for a correct operation.

Multi-Object Tracking evaluation

The last stage of the architecture to be qualitatively assessed is the Multi-Object Tracking stage. The correct behaviour of the algorithm can be checked if only one Bounding Box is displayed for a corresponding target for a long time in the simulator.

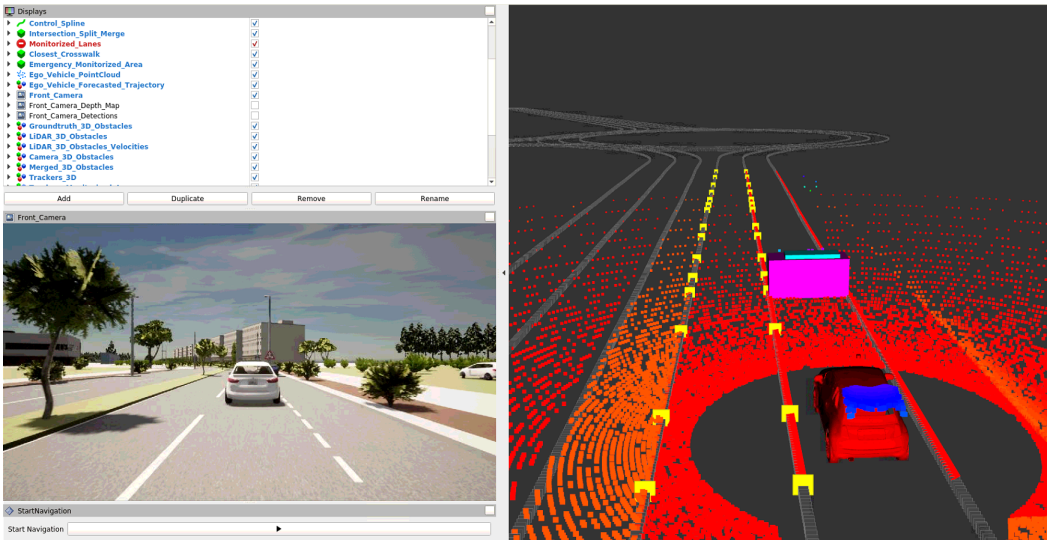


Figure 6.17: Sensor Fusion for dynamic vehicle pursuit on a straight track
Source. Own elaboration.

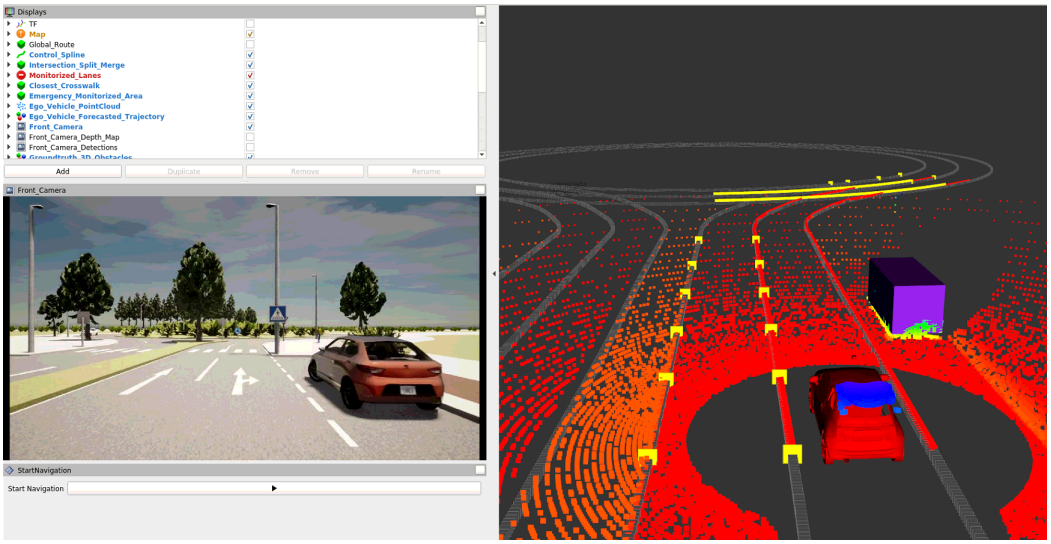


Figure 6.18: Multi-Object Tracking for stationary vehicle parked on the side
Source. Own elaboration.

The first scenario evaluated for Tracking depicted in Figure 6.18 shows a situation in which a vehicle parked on the left-hand side of the road is passed. The behaviour shown by the tracker is satisfactory, since the detections made on both point clouds are continuous in time and a Bounding Box representing the whole vehicle is displayed until the vehicle is lost from view by the sensors pointing to the front of the road. In addition, a constant reference size is provided by the ability to predict the position, as the Kalman Filter that drives the tracker is fed with the radar velocity, in this case, zero.

The second scenario to be shown explores the tracker's ability to track moving targets. In the situation shown in Figure 6.19 it is back to the roundabout where a car is approaching from the left of the ego vehicle. In this case, the tracker is able to make continuous detections of the vehicle until the pursuit begins in a more frontal manner. In that case, the Bounding Boxes that appear are seen in Figure 6.15, and doing the BEV-IoU of both detections becomes especially

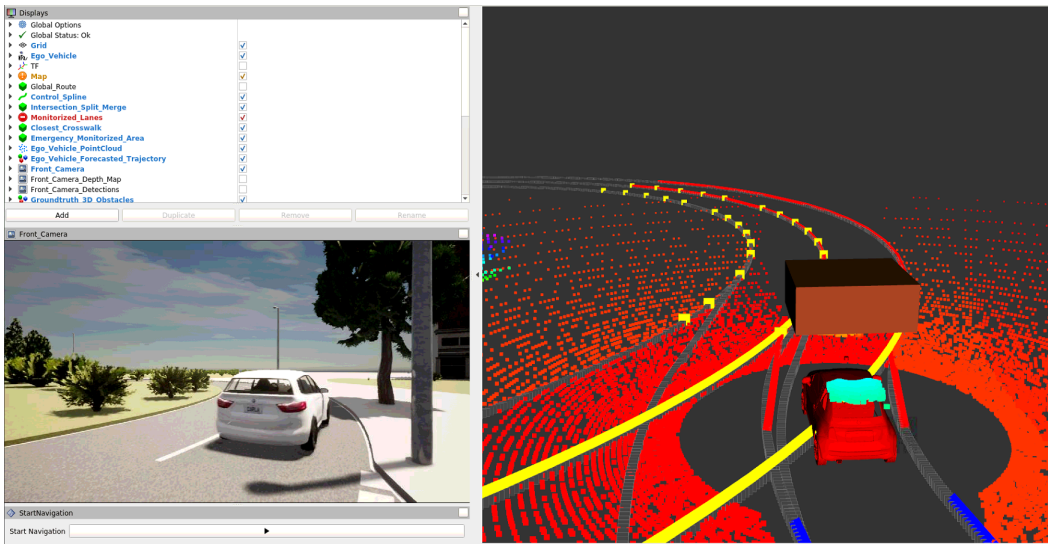


Figure 6.19: Multi-Object Tracking for dynamic vehicle pursuit at roundabout
Source. Own elaboration.

complicated. Still, the tracker is able to solve scenarios with static targets and simple scenarios with dynamic targets.

To continue with the evaluation of the Tracking system, it is proposed to implement a mechanism that allows to obtain a quantitative measure of the goodness of the Tracking system. From the information of the CARLA simulator, a set of data belonging to the vehicle information is obtained and will act as groundtruth. On the other hand, the data obtained through the implemented architecture is serialised in terms of position and velocity. Therefore, with both sets of data, we proceed to obtain a Root Mean Squared Error measurement for position and velocity and evaluate the system.

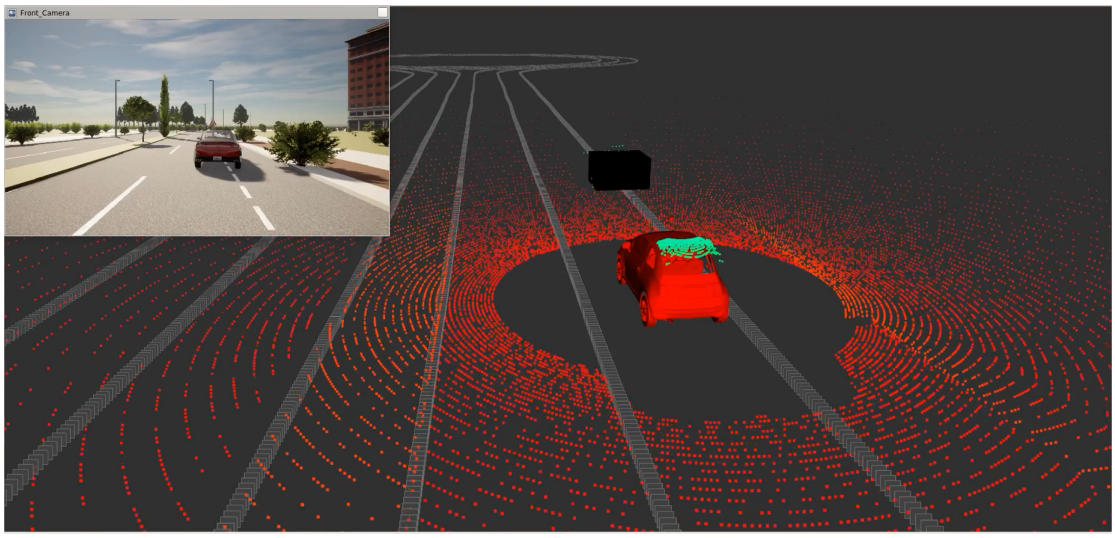


Figure 6.20: Scene for evaluation of Multi-Object Tracking
Source. Own elaboration.

Figure 6.20 presents the scenario where our application will be evaluated. A single vehicle moves through the scene making straight-line, diagonal and side-to-side movements in front of the ego vehicle. This represents a big challenge for the implemented application since the vehicle to be tracked will change its position, and this changes the way it should be clustered and detected. Figure 6.21 shows a bird's eye view of the trajectory of the vehicle during the scene. As a clarification for the reader, it is noted that the vehicle starts to move from the upper left corner of the figure. It can be seen that the position estimation is accurate enough to be considered successful. Much of the difference between the groundtruth dataset (blue) and the real dataset (orange) comes from the fact that the vehicle is clustered laterally and is not recognised with its full dimensions. Furthermore, it can be added that the groundtruth dataset is larger than the real dataset, so it can be deduced that the target has been lost sight of at the most difficult moments for the Detection and Tracking stages. However, calculating the RMSE for position, an error of 0.62 m in the X-axis and 1.25 m in the Y-axis is obtained. For more detailed information on the estimation error in position, images are attached to show the time evolution of the error for both axes, X and Y, in Figures 6.22 and 6.23.

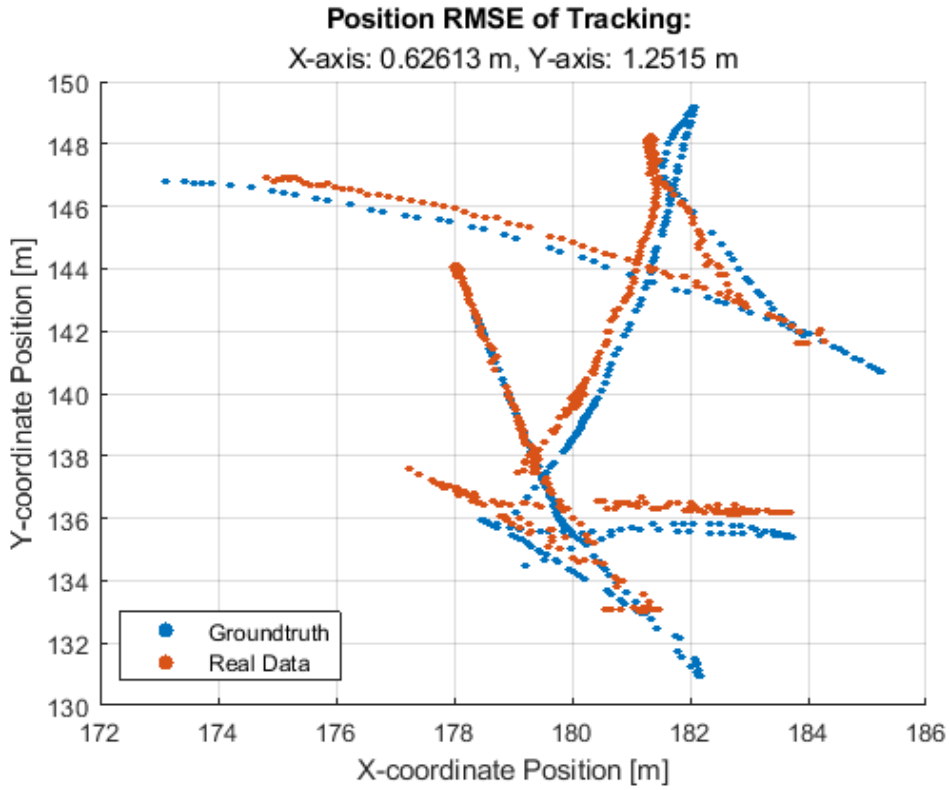


Figure 6.21: Position RMSE of Tracking for Single-Object Tracking
Source. Own elaboration.

In the case of velocity, a groundtruth dataset is taken from the information provided by the CARLA simulator for the vehicle magnitudes. To construct the real dataset, the relative Doppler velocity from the RADAR point cloud is taken and transformed to be referenced absolutely to the scene in which the figure is developed. Figure 6.24 plots the velocity of both datasets against the number of frames over which the sample is taken. Taking the RMSE of the linear velocity, an error result of 0.99 m/s is obtained.

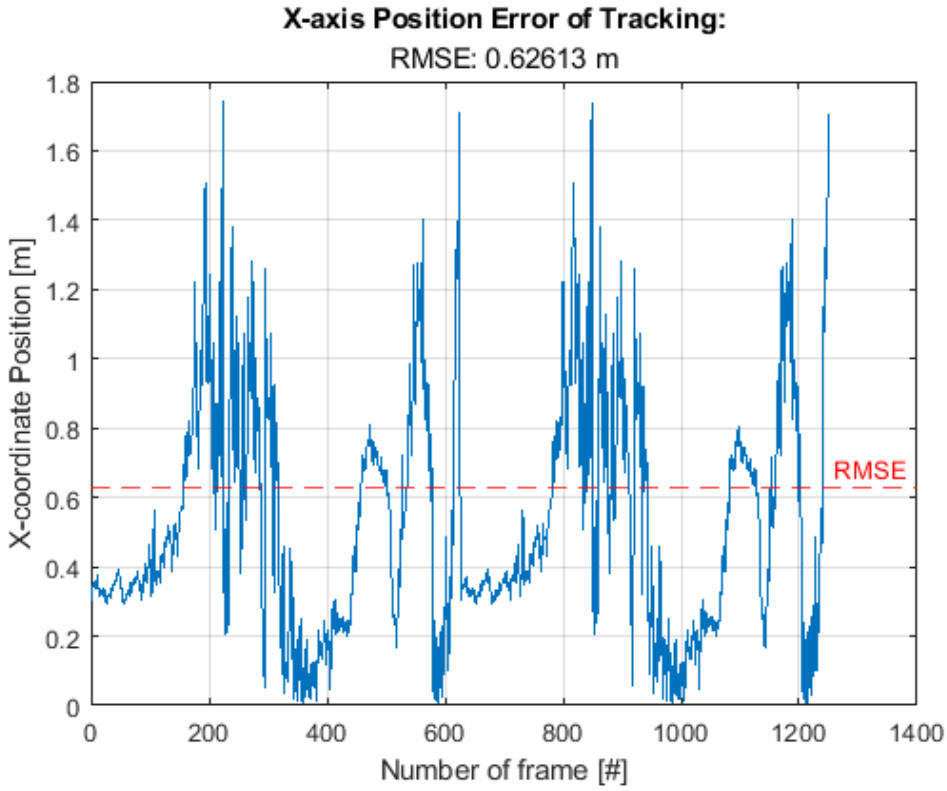


Figure 6.22: X-axis Position Error of Tracking for Single-Object Tracking Source. Own elaboration.

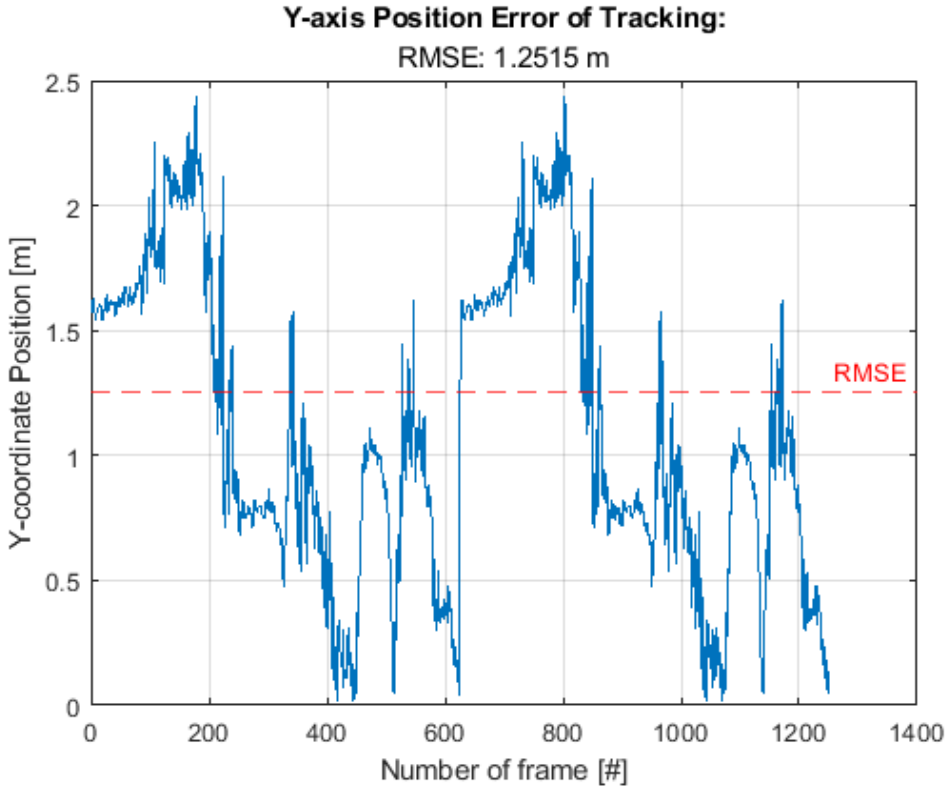


Figure 6.23: Y-axis Position Error of Tracking for Single-Object Tracking Source. Own elaboration.

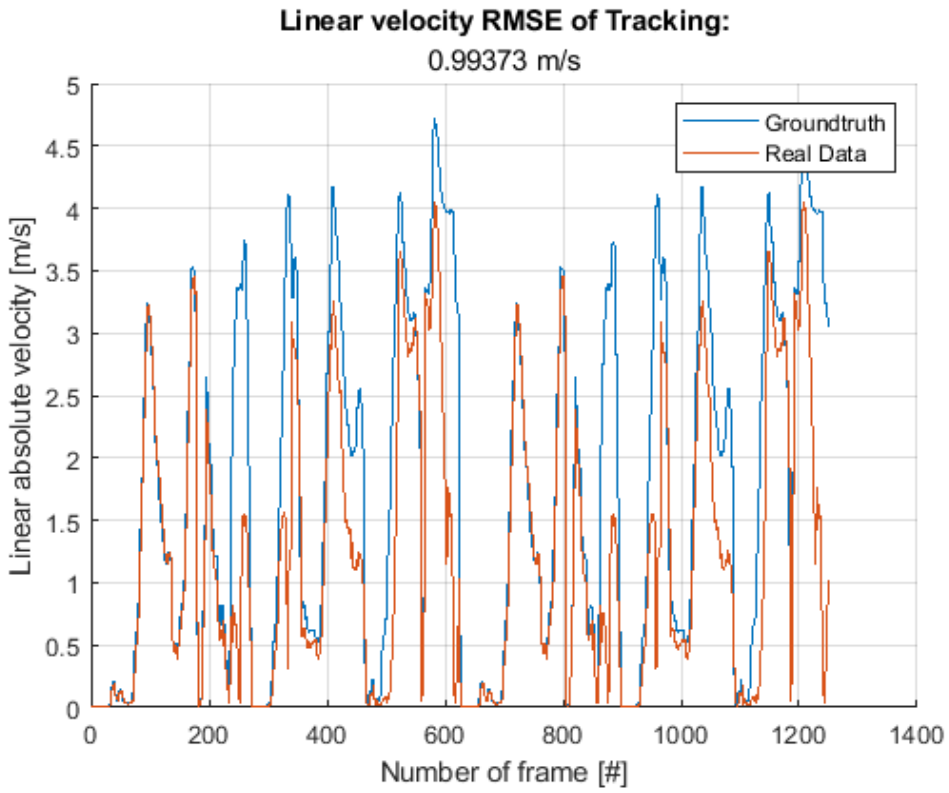


Figure 6.24: Linear Velocity RMSE of Tracking for Single-Object Tracking
Source. Own elaboration.

To conclude the analysis of the Tracking system, it is proceeded to obtain these mean square error magnitudes for position and velocity in a common use case in the autonomous driving research sector: an Adaptive Cruise Control scene. The scenario consists of the ego-vehicle pursuing a second vehicle and being able to maintain speed and/or distance with it. It is worth mentioning that for this simulation, the implemented system does not act on the vehicle's pedals or steering wheel and does not modify its speed, but only the performance of the perception system is of interest. This scenario is represented in Figure 6.25 and the positions of the ego-vehicle, the pursued vehicle, and the straight line on which the action takes place and which ends in a roundabout can be observed.

Figure 6.26 shows the performance of the application in terms of position. As in the previous case, the reader is reminded that the navigation starts in the upper left corner of the figure and ends in the lower right corner, where it is seen in the groundtruth dataset that the pursued vehicle starts to take the roundabout. When calculating the RMSE in position, an error of 0.20 m is obtained for the X-axis and an error of 0.65 m for the Y-axis. It is observed that the error decreases with respect to the previous scene. This occurs because the groudtruth centroid has been repositioned to the centre of the rear part of the car. The limitations resulting from not perceiving the full depth of the pursued vehicle are compensated by a centroid shift approximately equal to half the real length of the vehicle. This information on the estimation of position error is expanded upon two images that are attached to show the time evolution of the error for both axes, X and Y, in Figures 6.27 and 6.28.

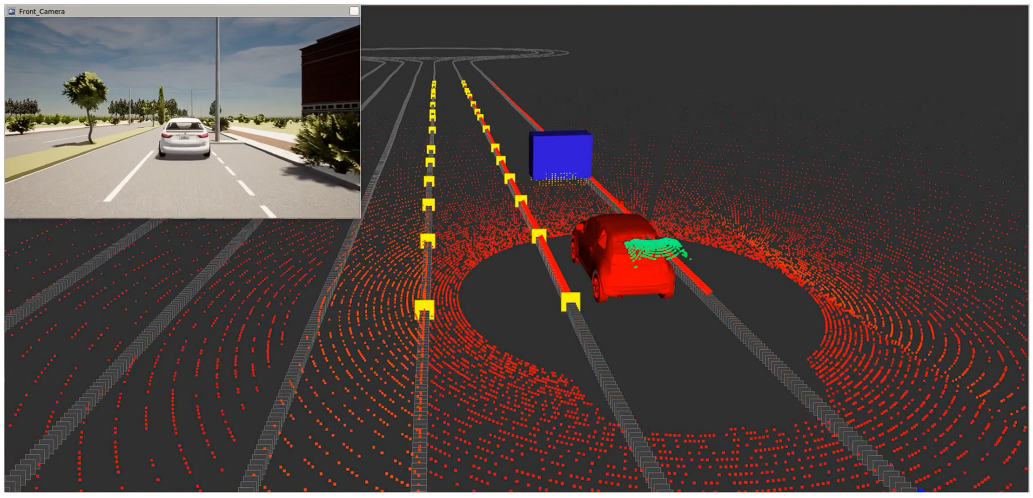


Figure 6.25: Scene for evaluation of ACC Use Case
Source. Own elaboration.

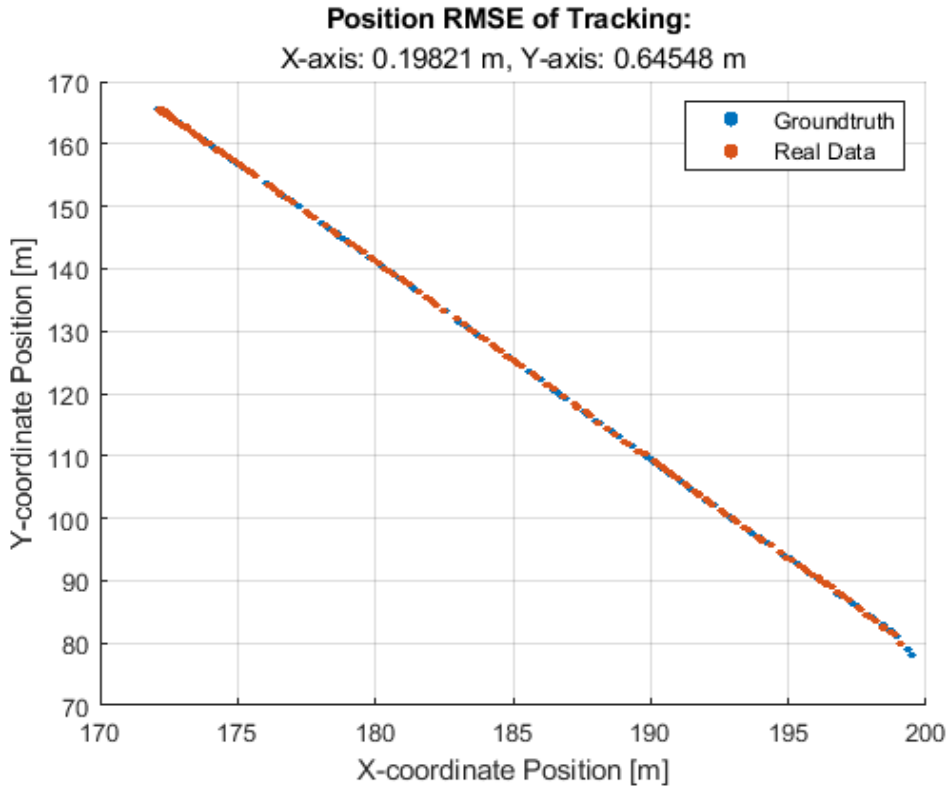


Figure 6.26: Position RMSE of Tracking for ACC Use Case
Source. Own elaboration.

As for the velocity analysis, this use case is a friendlier scenario than for the position. Although the position estimation is not as accurate as in the previous case due to the nature of these detections, they are continuous throughout the scenario, thus allowing a more faithful reproduction of reality to be obtained. This can be seen qualitatively in Figure 6.29, where the groundtruth data set and the real data set practically overlap. This theory is reinforced by the subsequent calculation of the linear velocity RMSE, where only an error of 0.2 m/s is obtained for a scenario where the maximum velocity is higher than in the previous case.

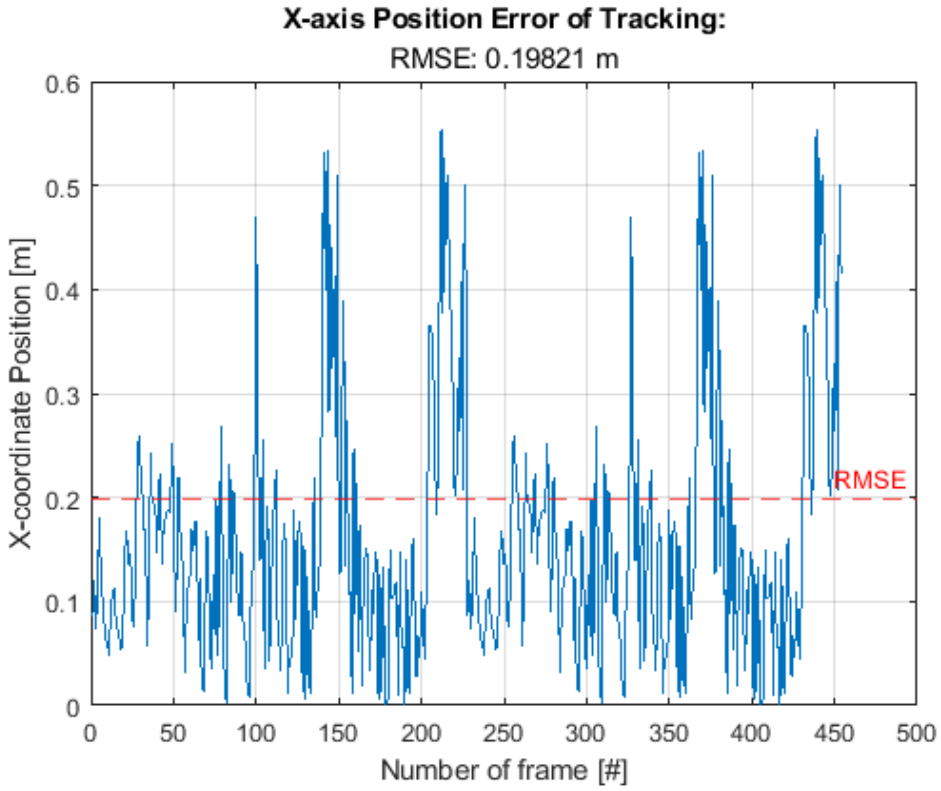


Figure 6.27: X-axis Position Error of Tracking for ACC Use Case Source. Own elaboration.

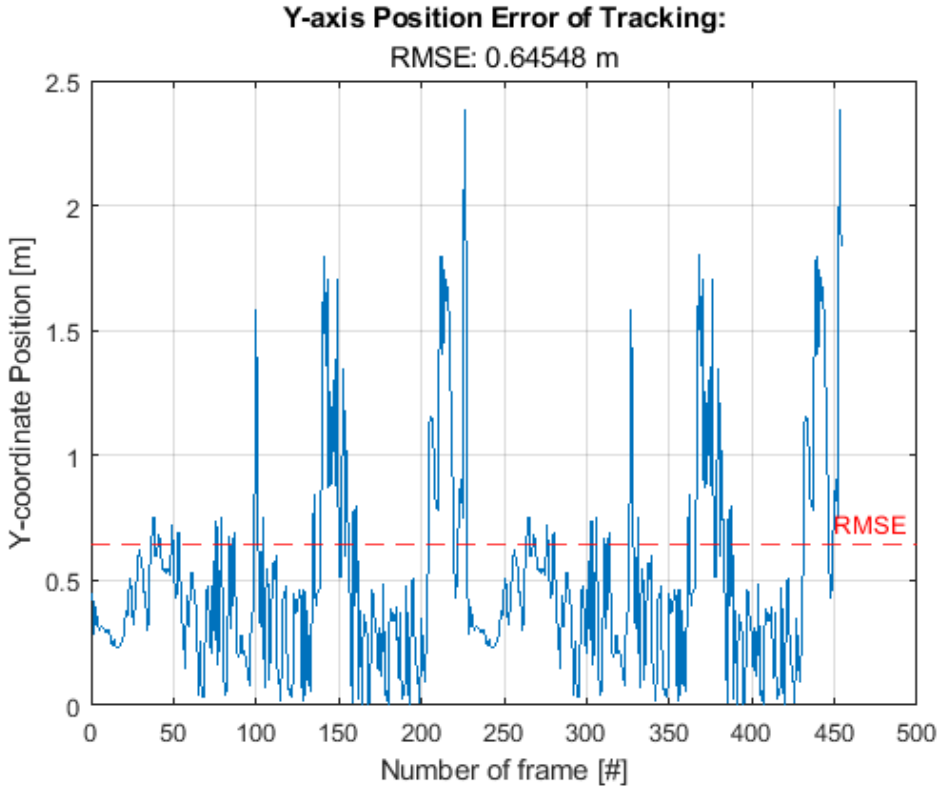


Figure 6.28: Y-axis Position Error of Tracking for ACC Use Case Source. Own elaboration.

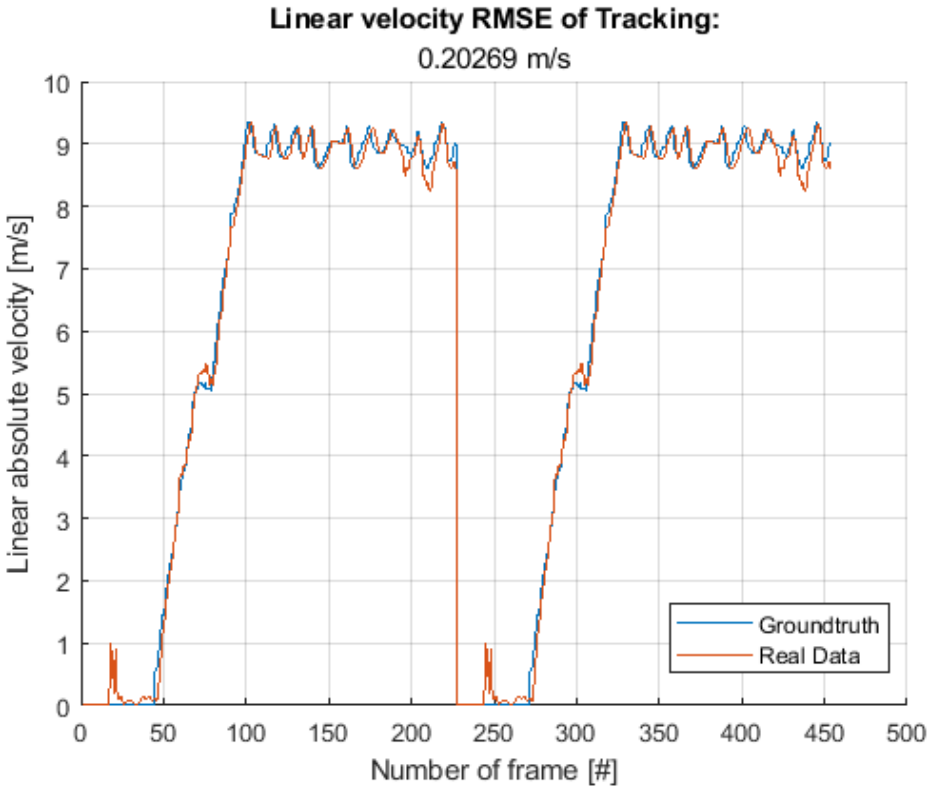


Figure 6.29: Linear Velocity RMSE of Tracking for ACC Use Case
Source. Own elaboration.

Time analysis

To conclude the high-level qualitative analysis, it is proceeded to highlight the real-time execution capability of the architecture. It is important to notice that this time analysis has been carried out on a personal computer whose characteristics are specified in Appendix A. Figure 6.30 shows the time consumption per frame of the algorithm. It is possible to observe that the highest processing time is required by the LiDAR point cloud for the Object Detection stage. This is due to the larger amount of points that populate this cloud, in addition to the need to include the RANSAC algorithm as processing prior to clustering. This effect is numerically verified in Table 6.6, which shows the exact breakdown of each and every task performed. Overall, the architecture has been shown to meet real-time requirements and to perform satisfactorily in simple and controlled environments.

Stage	Time (s)	Frequency (Hz)
LiDAR Object Detection	0.0196 s	51.02 Hz
RADAR Object Detection	0.0014 s	714.28 Hz
Sensor Fusion	0.0011 s	909.09 Hz
Multi-Object Tracking	0.0044 s	227.27 Hz
Total	0.0270 s	37.00 Hz

Table 6.6: Time analysis for the proposed architecture
Source. Own elaboration.

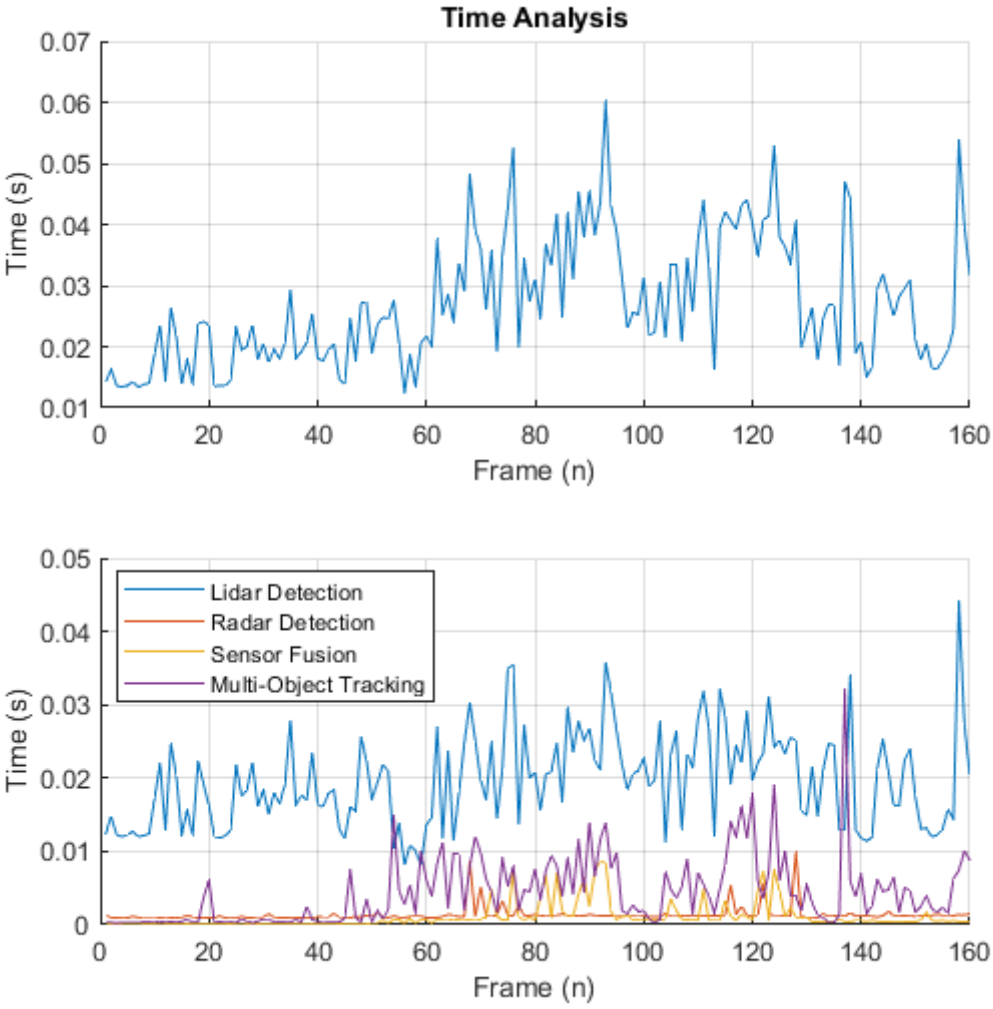


Figure 6.30: Time analysis for the proposed architecture
Source. Own elaboration.

Chapter 7

Conclusions and Future Works

It always seems impossible... until it is done.

Nelson Mandela

This chapter seeks to reflect on and draw conclusions from the work carried out over the last few months and reflected in this book. In addition, it is possible to discern lines of future work with which the work presented could be improved and with which the author could further deepen his knowledge in the area of Deep and Machine Learning applied to intelligent transport and autonomous driving technologies.

7.1 Conclusions

Recapitulating the entire path that has been travelled during this project, several conclusions can be drawn about the work done and the results obtained:

To reflect on the knowledge learned during the course of the work:

- As for the programming language used in the project, the author started with an average knowledge of the C programming language. In Python (which the author started with no previous knowledge), algorithms have been implemented by hand and advanced concepts from object-oriented programming and other more basic concepts from functional programming, such as filters, have been applied.
- In terms of the software technologies used during the project, different technologies that are at the cutting edge of their respective fields have been learned to be used. It should be borne in mind that the author started from scratch in many of them, such as the Carla simulator or Docker, and in the rest he started with initial knowledge, as was the case with ROS and Ubuntu. During the course of the project, the author has deepened his knowledge of them and after the end of this work, he has been able to put them into practice in professional environments.

Conclusions drawn from the theoretical work carried out:

- One of the theoretical studies carried out consisted of an in-depth study of the physical fundamentals governing LiDAR and RADAR sensors. By defining their main advantages and disadvantages, it was possible to understand the role played by each of them in a sensor fusion system.
- The state of the art in the fields of Object Detection, Object Tracking and Sensor Fusion has been studied in order to consolidate a contextualisation and introduction to the development of perception systems for intelligent vehicles, in which LiDAR, RADAR and camera sensors dominate with their successive fusions for the extraction of data from the scene.

Conclusions drawn from the practical development of the applications:

- A Multi-Object Tracking and Object Detection system for LiDAR and RADAR has been developed using processing libraries such as Open3D and Scikit-Learn. All this programmed in Python and using DBSCAN clustering techniques, data association for Sensor Fusion based on Intersection over Union 3D and an implementation of a Tracking algorithm based on SORT.
- Machine Learning-based and traditional Object Detection have some major limitations as the scenarios in which driving activities take place have a great diversity of events that need to be responded to. This is why state-of-the-art methods based on Deep Learning are used. However, the qualitative results for simple and controlled scenarios such as those used in this work are satisfactory.
- Both the state of the art study and the practical development of this work have shown that traditional tracking techniques work well and are still used today. However, they are dependent on a good performance of the Object Detection stage.

7.2 Future Works

To improve the work done during the development of the applications developed and reflected in the book.

- Development and implementation of an Object Detection system for Radar based on Doppler map information. In real-world applications, the information coming from Radar sensors is often given by Doppler maps in which the scene information is encoded. In this case, when working with the CARLA simulator, which provides point clouds as information from this sensor, an Object Detection method consistent with this data type is applied. In the case of implementing this system using traditional techniques, the use of one of the techniques of the CFAR family is recommended: CA-CFAR or OS-CFAR.

- To improve the prediction performance of the Tracker based on the SORT algorithm, the current Linear Kalman Filter could be replaced by an Unscented Kalman Filter that would be able to model non-linear dynamic equations in an efficient way. With these two improvements, it is obtained one of the most powerful versions of traditional tracking techniques available without using Deep Learning.
- Implementation of a quantitative metrics extraction system for the CARLA simulator. As the results presented in this work have only been qualitative, an improvement for the rigorousness of the evaluation of the applications would involve obtaining metrics such as those explained in Chapter 3 for the Object Detection and Multi-Object Tracking stages.

To expand the work developed in this book in terms of Sensor Fusion.

- Inclusion of the camera sensor. As seen in Chapter 3, a perception system increases its robustness to a larger amount of information received. The third sensor included in the triarchy of sensors that govern these systems in the state of the art today is the camera. Thanks to traditional and simple processing in accordance with the philosophy of this work, it would be possible to obtain semantic information of the road and an improvement of the realised Fusion.

To expand the work developed in this book by using Deep Learning techniques.

- Replacement of the existing SORT-based Tracking module with its Deep Learning-based predecessor, DeepSORT. The operating principles follow the philosophy of the original SORT but include neural networks that extract features from objects for improved results at the Tracking stage.
- Implementation of a state-of-the-art 3D Object Detection algorithm for point clouds. As seen in Chapter 3, the methods that currently top the detection rankings in benchmarks are based on Deep Learning methods for Detection. The logical improvement of this system would be to implement one of them including neural networks.

Bibliography

[1] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[2] L. M. Bergasa, J. Araluce, E. Romera, R. Barea, E. López-Guilén, J. del Egido, and C. A. Hernanz-Mayoral, "Naturalistic driving study for older drivers based on the drivesafe app," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2019, pp. 1574–1579.

[3] J. F. Arango, L. M. Bergasa, P. A. Revenga, R. Barea, E. López-Guillén, C. Gómez-Huélamo, J. Araluce, and R. Gutiérrez, "Drive-by-wire development process based on ros for an autonomous electric vehicle," *Sensors*, vol. 20, no. 21, p. 6121, 2020.

[4] M. Tradacete, Á. Sáez, J. F. Arango, C. G. Huélamo, P. Revenga, R. Barea, E. López-Guillén, and L. M. Bergasa, "Positioning system for an electric autonomous vehicle based on the fusion of multi-gnss rtk and odometry by using an extented kalman filter," in *Workshop of Physical Agents*. Springer, 2018, pp. 16–30.

[5] M. Hoshiya and E. Saito, "Structural identification by extended kalman filter," *Journal of engineering mechanics*, vol. 110, no. 12, pp. 1757–1770, 1984.

[6] R. Gutiérrez, E. López-Guillén, L. M. Bergasa, R. Barea, Ó. Pérez, C. Gómez-Huélamo, F. Arango, J. Del Egido, and J. López-Fernández, "A waypoint tracking controller for autonomous road vehicles using ros framework," *Sensors*, vol. 20, no. 14, p. 4062, 2020.

[7] C. Gómez-Huelamo, L. M. Bergasa, R. Barea, E. López-Guillén, F. Arango, and P. Sánchez, "Simulating use cases for the uah autonomous electric car," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2019, pp. 2305–2311.

[8] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "Pointpillars: Fast encoders for object detection from point clouds," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 12 697–12 705.

[9] C. Gómez-Huélamo, J. Del Egido, L. M. Bergasa, R. Barea, M. Ocana, F. Arango, and R. Gutiérrez-Moreno, "Real-time bird's eye view multi-object tracking system based on fast encoders for object detection," in *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2020, pp. 1–6.

[10] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[11] R. E. Kalman, "A new approach to linear filtering and prediction problems," 1960.

[12] A. Borkar, M. Hayes, and M. T. Smith, "Robust lane detection and tracking with ransac and kalman filter," in *2009 16th IEEE International Conference on Image Processing (ICIP)*. IEEE, 2009, pp. 3261–3264.

[13] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[14] E. Romera, J. M. Alvarez, L. M. Bergasa, and R. Arroyo, "Erfnet: Efficient residual factorized convnet for real-time semantic segmentation," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 1, pp. 263–272, 2017.

[15] L. M. Bergasa Pascual, Á. Saez Contreras, M. E. López Guillén, E. Romera Carmena, M. Tradacete Ágreda, C. Gómez Huélamo, J. d. Egido Sierra *et al.*, "Real-time semantic segmentation for fisheye urban driving images based on erfnet," 2019.

[16] J. Araluce, L. M. Bergasa, C. Gómez-Huélamo, R. Barea, E. López-Guillén, F. Arango, and Ó. Pérez-Gil, "Integrating openface 2.0 toolkit for driver attention estimation in challenging accidental scenarios," in *Workshop of Physical Agents*. Springer, 2020, pp. 274–288.

[17] T. Baltrusaitis, A. Zadeh, Y. C. Lim, and L.-P. Morency, "Openface 2.0: Facial behavior analysis toolkit," in *2018 13th IEEE international conference on automatic face & gesture recognition (FG 2018)*. IEEE, 2018, pp. 59–66.

[18] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, "A survey of autonomous driving: Common practices and emerging technologies," *IEEE access*, vol. 8, pp. 58 443–58 469, 2020.

[19] S. D. Pendleton, H. Andersen, X. Du, X. Shen, M. Meghjani, Y. H. Eng, D. Rus, and M. H. Ang, "Perception, planning, control, and coordination for autonomous vehicles," *Machines*, vol. 5, no. 1, p. 6, 2017.

[20] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[21] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2961–2969.

[22] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-decoder with atrous separable convolution for semantic image segmentation," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 801–818.

[23] Y. Yan, Y. Mao, and B. Li, "Second: Sparsely embedded convolutional detection," *Sensors*, vol. 18, no. 10, p. 3337, 2018.

[24] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[25] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*. Springer, 2016, pp. 21–37.

[26] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[27] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[28] P. Narksri, E. Takeuchi, Y. Ninomiya, Y. Morales, N. Akai, and N. Kawaguchi, "A slope-robust cascaded ground segmentation in 3d point cloud for autonomous vehicles," in *2018 21st International Conference on intelligent transportation systems (ITSC)*. IEEE, 2018, pp. 497–504.

[29] W. Hou, D. Li, C. Xu, H. Zhang, and T. Li, "An advanced k nearest neighbor classification algorithm based on kd-tree," in *2018 IEEE International Conference of Safety Produce Informatization (IICSPI)*. IEEE, 2018, pp. 902–905.

[30] E. Schubert, J. Sander, M. Ester, H. P. Kriegel, and X. Xu, "Dbscan revisited, revisited: why and how you should (still) use dbscan," *ACM Transactions on Database Systems (TODS)*, vol. 42, no. 3, pp. 1–21, 2017.

[31] E. Arnold, O. Y. Al-Jarrah, M. Dianati, S. Fallah, D. Oxtoby, and A. Mouzakitis, "A survey on 3d object detection methods for autonomous driving applications," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 10, pp. 3782–3795, 2019.

[32] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller, "Multi-view convolutional neural networks for 3d shape recognition," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 945–953.

[33] B. Li, T. Zhang, and T. Xia, "Vehicle detection from 3d lidar using fully convolutional network," *arXiv preprint arXiv:1608.07916*, 2016.

[34] B. Wu, A. Wan, X. Yue, and K. Keutzer, "Squeezeseg: Convolutional neural nets with recurrent crf for real-time road-object segmentation from 3d lidar point cloud," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 1887–1893.

[35] M. Simon, K. Amende, A. Kraus, J. Honer, T. Samann, H. Kaulbersch, S. Milz, and H. Michael Gross, "Complexer-yolo: Real-time 3d object detection and tracking on semantic point clouds," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2019, pp. 0–0.

[36] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, "Multi-view 3d object detection network for autonomous driving," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2017, pp. 1907–1915.

[37] J. Beltrán, C. Guindel, F. M. Moreno, D. Cruzado, F. Garcia, and A. De La Escalera, "Birdnet: a 3d object detection framework from lidar information," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2018, pp. 3517–3523.

[38] B. Yang, W. Luo, and R. Urtasun, "Pixor: Real-time 3d object detection from point clouds," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2018, pp. 7652–7660.

[39] B. Li, "3d fully convolutional network for vehicle detection in point cloud," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 1513–1518.

[40] M. Engelcke, D. Rao, D. Z. Wang, C. H. Tong, and I. Posner, "Vote3deep: Fast object detection in 3d point clouds using efficient convolutional neural networks," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 1355–1361.

[41] J. Deng, S. Shi, P. Li, W. Zhou, Y. Zhang, and H. Li, "Voxel r-cnn: Towards high performance voxel-based 3d object detection," *arXiv preprint arXiv:2012.15712*, 2020.

[42] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 652–660.

[43] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," *arXiv preprint arXiv:1706.02413*, 2017.

[44] Y. Zhou and O. Tuzel, "Voxelnet: End-to-end learning for point cloud based 3d object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4490–4499.

[45] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas, "Frustum pointnets for 3d object detection from rgb-d data," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 918–927.

[46] Z. Yang, Y. Sun, S. Liu, X. Shen, and J. Jia, "Ipod: Intensive point-based object detector for point cloud," *arXiv preprint arXiv:1812.05276*, 2018.

[47] S. Shi, X. Wang, and H. Li, "Pointrcnn: 3d object proposal generation and detection from point cloud," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 770–779.

[48] W. Zheng, W. Tang, L. Jiang, and C.-W. Fu, "Se-ssd: Self-ensembling single-stage object detector from point cloud," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 14 494–14 503.

[49] Z. Wang, Y. Wu, and Q. Niu, "Multi-sensor fusion in automated driving: A survey," *IEEE Access*, vol. 8, pp. 2847–2868, 2019.

[50] W. Elmenreich, "An introduction to sensor fusion," *Vienna University of Technology, Austria*, vol. 502, pp. 1–28, 2002.

[51] T. Yin, X. Zhou, and P. Krahenbuhl, "Center-based 3d object detection and tracking," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 11 784–11 793.

[52] S. Vora, A. H. Lang, B. Helou, and O. Beijbom, "Pointpainting: Sequential fusion for 3d object detection," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 4604–4612.

[53] S. Pang, D. Morris, and H. Radha, "Clocs: Camera-lidar object candidates fusion for 3d object detection," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 10 386–10 393.

[54] D. Frossard and R. Urtasun, "End-to-end learning of multi-sensor 3d tracking by detection," in *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 635–642.

[55] X. Weng, J. Wang, D. Held, and K. Kitani, "Ab3dmot: A baseline for 3d multi-object tracking and new evaluation metrics," *arXiv preprint arXiv:2008.08063*, 2020.

[56] X. Weng, Y. Wang, Y. Man, and K. M. Kitani, "Gnn3dmot: Graph neural network for 3d multi-object tracking with 2d-3d multi-feature learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 6499–6508.

[57] P. Voigtlaender, M. Krause, A. Osep, J. Luiten, B. B. G. Sekar, A. Geiger, and B. Leibe, "Mots: Multi-object tracking and segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 7942–7951.

[58] A. Kim, A. Ošep, and L. Leal-Taixé, "Eagermot: 3d multi-object tracking via sensor fusion," *arXiv preprint arXiv:2104.14682*, 2021.

[59] A. Milan, L. Leal-Taixé, I. Reid, S. Roth, and K. Schindler, "Mot16: A benchmark for multi-object tracking," *arXiv preprint arXiv:1603.00831*, 2016.

[60] J. Luiten, A. Osep, P. Dendorfer, P. Torr, A. Geiger, L. Leal-Taixé, and B. Leibe, "Hota: A higher order metric for evaluating multi-object tracking," *International Journal of Computer Vision*, pp. 1–31, 2020.

[61] Y. Li and J. Ibanez-Guzman, "Lidar for autonomous driving: The principles, challenges, and trends for automotive lidar and perception systems," *IEEE Signal Processing Magazine*, vol. 37, no. 4, pp. 50–61, 2020.

[62] I. Bilik, O. Longman, S. Villeval, and J. Tabrikian, "The rise of radar for autonomous vehicles: Signal processing solutions and future research directions," *IEEE Signal Processing Magazine*, vol. 36, no. 5, pp. 20–31, 2019.

[63] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[64] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.

[65] D. Merkel *et al.*, "Docker: lightweight linux containers for consistent development and deployment," *Linux journal*, vol. 2014, no. 239, p. 2, 2014.

[66] Q.-Y. Zhou, J. Park, and V. Koltun, "Open3D: A modern library for 3D data processing," *arXiv:1801.09847*, 2018.

[67] S. Lloyd, "Least squares quantization in pcm," *IEEE transactions on information theory*, vol. 28, no. 2, pp. 129–137, 1982.

[68] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *kdd*, vol. 96, no. 34, 1996, pp. 226–231.

[69] H. Steinhaus *et al.*, "Sur la division des corps matériels en parties," *Bull. Acad. Polon. Sci*, vol. 1, no. 804, p. 801, 1956.

[70] B. Yang, R. Guo, M. Liang, S. Casas, and R. Urtasun, "Radarnet: Exploiting radar for robust perception of dynamic objects," in *European Conference on Computer Vision*. Springer, 2020, pp. 496–512.

[71] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple online and realtime tracking," in *2016 IEEE International Conference on Image Processing (ICIP)*, 2016, pp. 3464–3468.

Appendix A

Specifications

This appendix includes in detail all the hardware and software resources that have been necessary to carry out this Final Degree Project.

A.1 Hardware resouces

- Desktop Personal Computer with the following specifications:
  - Intel i7-7700 processor @ 4.2GHz or similar.
  - RAM DDR5 24GB or similar.
  - AMD RX580 Nitro 8GB graphics card or similar.
  - Solid state hard disk of at least 500GB.

A.2 Software resources

- Ubuntu 18.04 LTS and 20.04 operating systems.
- CARLA hyper-realistic autonomous navigation simulator, version 0.9.11.
- Visual Studio Code programming environment for Python language.
- Python frameworks: Scikit-Learn, Open3D, Numpy.
- ROS framework for robotics software development.
- Docker containerization software.
- Git control version software.
- TexMaker programming environment for documentation purposes in LaTeX.

Appendix B

Budget

This appendix contains a budget for the implementation of the project, which is composed of the following items:

- Cost for the use of software and hardware materials involved in the project.
- Personnel costs for each of the major sub-tasks into which the project has been divided.
- Total costs including applicable Spanish taxes (21%).

B.1 Hardware and Software Costs

As for the hardware resources, a personal desktop computer is used for the whole project, both for the theoretical study and the state of the art and for the development of the applications and the writing of the memory. The software resources have been chosen following an open-source philosophy, so that access to all the tools is completely free of charge.

	CONCEPT	**PRIZE**
Hardware	Desktop Personal Computer	1,400.00 €
Software	Ubuntu Operating System	0.00 €
	CARLA Simulator	0.00 €
	Visual Studio Code	0.00 €
	Robot Operating System	0.00 €
	Docker	0.00 €
	Git	0.00 €
	TexMaker	0.00 €
	TOTAL PRIZE	1,400.00 €

Table B.1: Breakdown of Hardware and Software usage costs
Source. Own elaboration

B.2 Personnel Costs

As for the cost of the work carried out by the author during the project, an average of 4 hours of work per day has been taken for the whole project. This amount has varied during the year, as there are stages in which he has focused more on the subjects of the degree and other stages in which he has dedicated himself full time to the development of this work.

CONCEPT	HOURLY COST	TIME	PRIZE
Theoretical study of traditional techniques for 3D Object Detection and Multi-Object Tracking	15 €/h.	3 months	3,960.00 €
Study of the state of the art of 3D Object Detection and Multi-Object Tracking	20 €/h.	1 month	1,760.00 €
Development of Multi-Object Tracking System based on LiDAR and RADAR	15 €/h.	3 months	3,960.00 €
Documentation and writing of the Final Degree Project book	12 €/h.	2 months	2,112.00 €
TOTAL PRIZE			11,792.00€

Table B.2: Breakdown of Personnel costs
Source. Own elaboration

All tasks have been assessed taking into account the level of difficulty and the role he should have in a standard technology company, as he has gone through stages where the author can be considered as a technician, a developer or a researcher.

B.3 Total Costs

Once all the costs involved in the implementation of the project have been broken down, the total cost of the project is calculated.

CONCEPT	PRIZE
Hardware and Software Costs	1,400.00 €
Personnel Costs	11,792.00 €
VAT (21%)	2,770.32 €
TOTAL PRIZE	15,962.32 €

Table B.3: Breakdown of Total costs
Source. Own elaboration

The project amounts to the sum of FIFTEEN THOUSAND NINE HUNDRED AND SIXTY TWO EUROS AND THIRTY TWO CENTS (15,962.32 EUROS), which includes the total costs of material and personnel plus the corresponding Spanish taxes (21%), for a total duration of 9 months and presented in September 2021.

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR

Universidad
de Alcalá